M. David Johnson
http://www.bds-soft.com
info@bds-soft.com

Back To

(Almost)

Bare-Metal Programming

by M. David Johnson

2021/09/29

Abstract

Going back to (almost) bare metal, The ML Foundation Core is presented as the beginning of a system for simplifying and organizing the development of 6809 Assembly Language Programs and Systems for the Radio Shack Color Computer; most specifically in the areas of non-graphic numerical computation and text processing, and primarily for the 64K CoCo 2.

——

This paper and its associated code are available online at:

http://www.bds-soft.com/cocoPapers.php .

=====

Table of Contents

=====

Introduction

From Techopedia:

> Bare-metal programming is a term for programming that operates without various layers of abstraction or, as some experts describe it, "without an operating system supporting it." Bare-metal programming interacts with a system at the hardware level, taking into account the specific build of the hardware.

——

In this paper, I present the beginnings of the Core of The ML Foundation System, a system which will provide the bedrock upon which you can build more extensive Assembly Language programs and systems of your own.

When I first started learning 6809 Assembly Language back in the late 1980’s, I found it to be quite a struggle. Books by experts such as Bill Barden, Lance Leventhal, and others were significant helps, but getting past the first hurdle of being able to put information into the computer and get stuff back out was daunting, to say the least.

With the completion of this paper, you have a minimally complete system; capable of receiving input from the Keyboard and generating output to the Video RAM (**`VIDRAM`**) Screen.

Please note that this system **IS NOT COMPLETE** and **IT WILL CHANGE** !

It’s enough for you to begin playing and experimenting with, but please note that anything built on this minimal system will probably get broken by future developments.

The ML Foundation is also intended to ultimately be part of the VCC Bundle described in (MDJ02), and future developments will be guided by that intention.

——

Back in 1985, James and Victor Perotti wrote:

> There is no better way to learn about computers than by learning to program in Assembly. With it you are directly manipulating the CPU ; you are writing in the language of the machine; you are learning how the computer works. With other languages, you program in the environment of that particular software. Assembly is the lowest level language, the one closest to the raw binary code that the CPU really processes. (Perotti 68)

And more recently, Ed Snider writes, “In programming the Color Computer, my language of choice is assembly language. It’s the only way to get full performance out of the machine, and to control the hardware directly.”

——

In order to build anything, including software, you need a strong and solid foundation on which to build (cf. what Jesus said about the house that was built on sand: Matthew 7:24-27). What I hope to provide here is a solid but simple foundation; one which is easy to understand and won’t distract you by sending you down a rabbit hole, chasing after the true meaning of a tricky bit of code.

Efficiency is good, but clarity is better.

Consider, for a moment, the following BASIC program.

```
1DATA182,28,0,31,137,61,253,28,
0,57:PCLEAR1:CLEAR200,&H1C00:FO
RI=0TO9:READN1:POKE&H6000+I,N1:
NEXTI
2INPUTN:N=INT(ABS(N)):IFN>255TH
ENN=255
3POKE&H1C00,N:EXEC&H6000:N2=(((
PEEK(&H1C00))*256)+PEEK(&H1C01)
):?N;"**2=";N2:?"MORE(Y/N)";:IN
PUTZ$:IFZ$="Y"GOTO2:END
```

DON’T TURN THE PAGE YET !

This three-line program is named SB.BAS —

Can you tell what this program is supposed to do?

And, if you can tell what it’s supposed to do, can you tell how it goes about doing it?

THREE… HOURS… LATER…

Okay, now you can turn the page.

——

This is the same program, only not quite so crunched up.

```
1'SQRBYTSV.BAS
2'E.G. SQRBYT SEMI-VERBOSE
3'MDJ 2021/09/13
4PCLEAR1
5CLEAR200,&H1C00
6DATA182,28,0,31,137,61,253,28,0,57
7FORI=0TO9
8READN1
9POKE&H6000+I,N1
10NEXTI
11INPUTN
12N=INT(ABS(N))
13IFN>255THENN=255
14POKE&H1C00,N
15EXEC&H6000
16N2=(((PEEK(&H1C00))*256)+PEEK(&H1C01))
17PRINTN;"**2=";N2
18PRINT"MORE(Y/N)";
19INPUTZ$
20IFZ$="Y"GOTO11
21END
```

Is that any better?

"Not much," you say.

Okay, go on to the next page.

———

This is essentially the same program again, but now fully commented and with some better direction for the I/O tasks:

```
1000 '*****
1010 '*
1020 '* SQRBYTV.BAS
1030 '* I.E. SQRBYT VERBOSE
1040 '* MDJ 2021/09/13
1050 '*
1060 '* SQUARES AN
1070 '* UNSIGNED BYTE
1080 '* IN MACHINE CODE
1090 '*
1100 '*****
1110 '
1120 'SETUP MEMORY
1130 PCLEAR 1
1140 CLEAR 200, &H1C00
1150 '
1160 'THE MACHINE LANGUAGE
1170 'ROUTINE FROM SQRBYT.ASM
1180 DATA 182,28,0,31,137
1190 DATA 61,253,28,0,57
1200 '
1210 'PUT THE MACHINE LANGUAGE
1220 'PROGRAM INTO MEMORY
1230 FOR I = 0 TO 9
1240 READ N1
1250 POKE &H6000+I,N1
1260 NEXT I
1270 '
1280 'UNCOMMENT FOR DEBUGGING
1290 'PRINT
1300 'FOR I = 0 TO 9
1310 'N = PEEK(&H6000+I)
1320 'PRINT N;",";
1330 'NEXT I
1340 'PRINT
1350 '
1360 'ENTER A NUMBER
1370 A$="ENTER AN UNSIGNED "
1380 B$="BYTE: "
1390 PRINT A$;B$;
1400 INPUT N
1410 '
1420 'NO NEGATIVE NUMBERS
1430 N=ABS(N)
```

```
1440 '
1450 'ONLY UNSIGNED INTEGERS
1460 N=INT(N)
1470 '
1480 'MAXIMUM SIZE = 8-BITS
1490 IF N>255 THEN N=255
1500 '
1510 'PUT N TO TRANSFER REGA
1520 POKE &H1C00,N
1530 '
1540 'GO DO THE SQUARE
1550 EXEC &H6000
1560 '
1570 'GET THE SQUARE FROM
1580 'TRANSFER REGD
1590 NA=PEEK(&H1C00)
1600 '
1610 'UNCOMMENT FOR DEBUGGING
1620 'PRINT"NA = ";NA
1630 '
1640 NB=PEEK(&H1C01)
1650 '
1660 'UNCOMMENT FOR DEBUGGING
1670 'PRINT"NB = ";NB
1680 '
1690 N2=((NA * 256) + NB)
1700 '
1710 'REPORT THE RESULTS
1720 A$="THE SQUARE OF "
1730 B$=" IS "
1740 PRINT A$;N;B$;N2
1750 '
1760 'DO IT AGAIN?
1770 PRINT
1780 PRINT "DO ANOTHER (Y/N)?"
1790 INPUT Z$
1800 PRINT
1810 IF Z$="Y" GOTO 1370
32767 END
```

Now, you should pretty much be able to tell what the program does at a glance. But, it's still difficult to tell HOW it does it, until we add the associated Assembly Language Routine which is provided on the following page.

BTW, the debugging statements are not just included as eye-candy. I originally had a typo in Line 1190, having typed in a "38" instead of a "28".

At this point, everything should be clear.

```
                    00100 *****
                    00110 *
                    00120 * SQRBYT.ASM
                    00130 * MDJ 2021/09/13
                    00140 *
                    00150 * SQUARES AN
                    00160 * UNSIGNED BYTE
                    00170 *
                    00180 * ENTRY CONDITIONS:
                    00190 *   A = THE BYTE
                    00200 *
                    00210 * EXIT CONDITIONS:
                    00220 *   D = THE SQUARE
                    00230 *
                    00240 *****
                    00250
                    00260 * 8-BIT TRANSFER
                    00270 * REGISTER A
          1C00      00280 REGA    EQU     $1C00
                    00290
                    00300 * 16-BIT TRANSFER
                    00310 * REGISTER D
          1C00      00320 REGD    EQU     $1C00
                    00330
                    00340 * A = HIGH BYTE OF D
                    00350 * THUS THEY HAVE THE
                    00360 * SAME ADDRESS
                    00370
6000                00380         ORG     $6000
                    00390
                    00400 * GET THE BYTE
6000 B6   1C00      00410         LDA     REGA
                    00420
                    00430 * COPY IT TO REGISTER B
6003 1F   89        00440         TFR     A,B
                    00450
                    00460 * DO THE SQUARING
6005 3D             00470         MUL
                    00480
                    00490 * PUT THE 16-BIT RESULT
6006 FD   1C00      00500         STD     REGD
                    00510
                    00520 * EXIT
6009 39             00530         RTS
          0000      32767         END
```

00000 TOTAL ERRORS

Note that this code consists of ten bytes:

$B6, $1C, $00, $1F, $89, $3D, $FD, $1C, $00, $39

whose decimal equivalents are:

182, 028, 000, 031, 137, 061, 253, 028, 000, 057

or more compactly:

182,28,0,31,137,61,253,28,0,57

which represents the actual machine language code equivalent of the above assembly language routine.

——

There are times when super-crunched code like the original 3-Liner above is appropriate. Like when you're entering a CoCo-Stuffing contest. But, even then, you'd be wise to have a well-commented copy on hand somewhere.

In such cases, I like to keep a simple Microsoft Access database table on my Big Iron (Dell Intel i7-9700, 3.00 GHz, 32.0 GB RAM, 1.0 TB SDD, 2x4.0 TB HDD, Windows 10 Pro 64-bit) which includes:

1. A 12-character text field for the 8.3 crunched filename,
2. A 36-character text field for a brief description,
3. A 12-character text field for the 8.3 fully commented BASIC filename,
4. A 12-character text field for the 8.3 Assembly Language filename,
5. A 255-character text field for the first (up to) 255 characters of the text of the
6. A memo field of effectively unlimited length.

What I hope to provide here, and in following papers, is a strong, solid, and simple Color Computer machine language foundation ("The ML Foundation") upon which you can build systems of your own. This ML Foundation will be well-commented throughout — not only for you, but also for me when I come back to it and try to understand what I was doing five months (five minutes?) ago.

Others can, and have, written fancier and faster code than I can. But my goal is to provide you with an easy to use place to begin. Fast and fancy can come later.

Now, even a foundation needs a foundation. When you build a house, the first thing you build is its foundation. But even before you start laying that foundation, you make sure you've dug down to solid rock.

This initial paper presents the beginnings of the foundation of the ML Foundation, i.e. the ML Foundation Core. The ML Foundation is not completed in this paper - there remains much yet to be done. Even the Core itself is not completed here, but only what might be called the core of the Core. Yet to be added are Interrupt Routines, access to ROM LOADM and SAVEM code, and other such low-level stuff.

As Winston Churchill remarked at the Lord Mayor's dinner at Mansion House in London on November 10, 1942, just after the victory at El Alamein, "Now this is not the end. It is not even the beginning of the end. But it is, perhaps, the end of the beginning." (Manchester and Reid 591).

——

A Note on Numbers: To keep everything simple to understand, and also neatly lined-up, I frequently refer to numbers as decimal bytes with three full digits, e.g. 004, 027, 229, etc. See Appendix A for conversions between the decimal and hexadecimal representations of bytes.

In works of this complexity (at least for me) typos and other errors are bound to sneak in. Please let me know about any you discover so I can note and correct them.

M.D.J. 2021/09/29
info@bds-soft.com

=====

General Methodology

I'm developing The ML Foundation in two parts:

1. The ML Foundation Core, which will include the lowest level code, and which will be established in specific, unalterable, locations in Low RAM.

2. The rest of The ML Foundation, which will be relocatable (position-independent) code which, along with your own code if you also develop it to be relocatable, can be moved to anywhere in unallocated RAM at your convenience.

One caution regarding relocatable code: you have to remember where you put it so other routines can call it properly at its current address. In collecting all the parts of The ML Foundation, I'll generally specify preferred locations for all routines, even though they're written to be relocatable.

I'm also developing The ML Foundation, even within the Core, to be modular, rather than monolithic. This will allow each routine to be developed, written, and tested independently, greatly (hopefully) reducing both development and debugging time.

For each of the following Core Routines, I present the Assembly Language Routine itself, a Test Routine also written in Assembly Language, a BASIC Language Control Program to initiate and drive the Test Routine, and the results of the testing.

=====

MC6809 Register Set

The Register Set of the MC6809 is presented graphically on the Programmer's Card on the following page; from (Warren 154).

The X, Y, U, S, and PC Registers are each 16-bits wide.

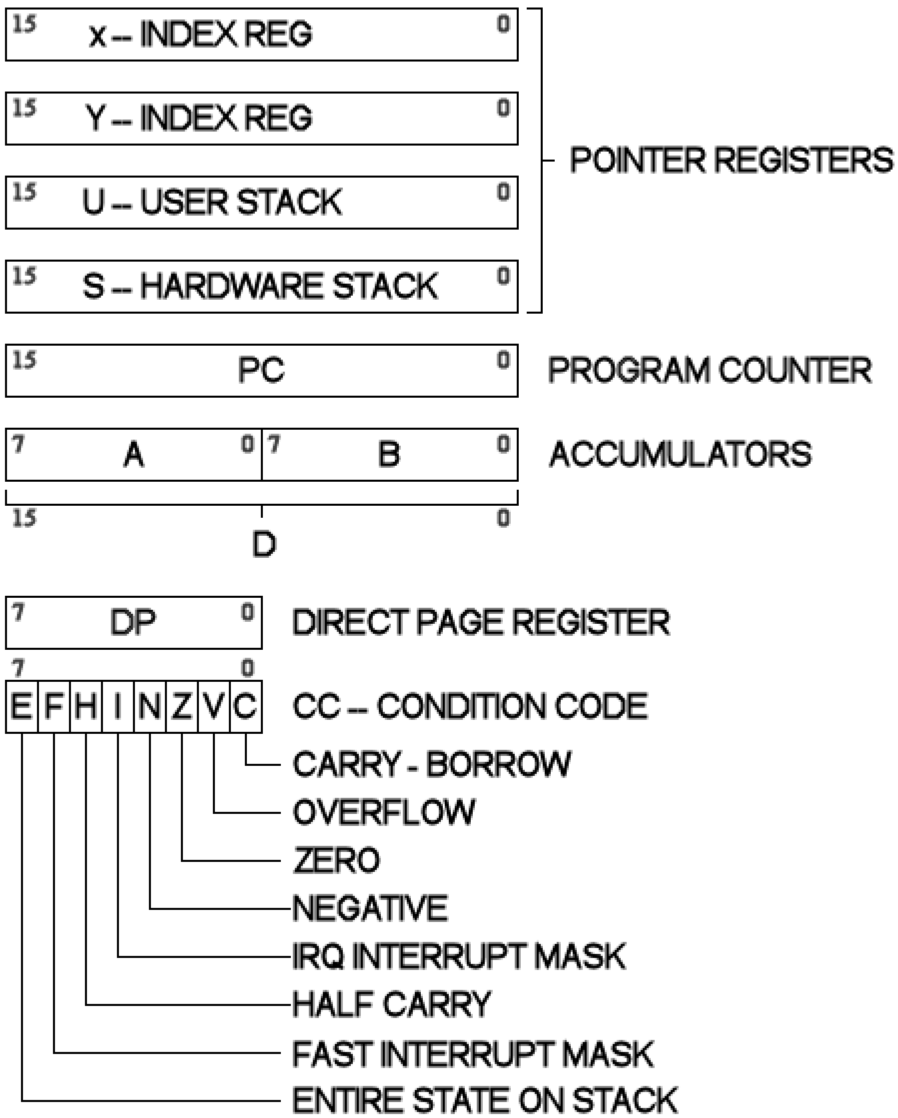
The A, B, DP, and CC Registers are each 8-bits wide.

The A and B Registers also combine to form the 16-bit D Register, with the A Register serving as the D Register's High Byte, and the B Register serving as its Low Byte.

In the 16-bit Registers, as shown on the Programmer's Card, the bits are numbered from 0 to 15 from right-to-left.

The bits in the 8-bit Registers are numbered from 0 to 7 from right-to-left.

Programmer's Card



From MC6809 Cookbook, page 154

=====

64K CoCo 2 Memory Map
“Out of the Box”

Here’s the 64K CoCo 2 Memory Map from page 254 of Getting Started With Extended Color BASIC (after correction of some typographical errors):

```
64K CoCo 2 General Memory Map
=============================

Decimal         Address Contents    Hex Address
-------         ----------------    -----------

                SYSTEM RAM:

0-1023          System Use          0000-03FF

1024-1535       Text Screen Memory  0400-05FF

                Graphic
                Screen Memory
1536-3071       Page 1              0600-0BFF
3072-4607       Page 2              0C00-11FF
4608-6143       Page 3              1200-17FF
6144-7679       Page 4              1800-1DFF
7680-9215       Page 5              1E00-23FF
9216-10751      Page 6              2400-29FF
10752-12287     Page 7              2A00-2FFF
12288-13823     Page 8              3000-35FF

                Program and
13824-32767     Variable Storage    3600-7FFF

                SYSTEM ROM:

                Extended
32768-40959     Color BASIC         8000-9FFF

40960-49151     Color BASIC         A000-BFFF

                Disk Basic or
49152-57343     Cartridge Memory    C000-DFFF

                Super-Extended
57344-65279     (Enhanced) Basic    E000-FEFF
```

	Hardware Registers, I/O, and Interrupt Vectors:	
65280-65535	Registers & Vectors	FF00-FFFF

=====

64K CoCo 2 Memory Map
ML Foundation Development

Here’s the modified “ML Foundation Development” Memory Map which I plan to use while developing this system:

```
64K CoCo 2 ML Foundation Development Memory Map
===============================================

Decimal         Address Contents    Hex Address
-------         ----------------    -----------

                SYSTEM RAM:

0-1023          System Use          0000-03FF

1024-1535       Text Screen Memory  0400-05FF

                UNUSED:
                Graphic Screen
1536-3071       Page 1              0600-0BFF

                BASIC Language
3072-7167       Program Storage     0C00-1BFF

                Assembly Language
7168-32767      Program Storage     1C00-7FFF

                SYSTEM ROM:

                Extended
32768-40959     Color BASIC         8000-9FFF

40960-49151     Color BASIC         A000-BFFF

                Disk Basic or
49152-57343     Cartridge Memory    C000-DFFF

                Super-Extended
57344-65279     (Enhanced) Basic    E000-FEFF
```

	Hardware Registers, I/O, and Interrupt Vectors:	
65280-65535	Registers & Vectors	FF00-FFFF

____

I'm leaving Graphic Page 1 unused because, during initial development, I don't plan on doing anything with graphics.

Also, during the earliest part of this development project, I discovered that stuff I put into Graphic Page 1 memory, even after "PCLEAR 0", often became corrupted for no reason I could easily discern.

(Phooey!)

I also plan to use the following Low RAM "unused" memory addresses as scratchpad variables wherever needed:

```
$0076 - $0077     2 bytes
$00F3 - $00FF    13 bytes
```

Scratchpad variables are defined as being restricted to within a given isolated routine, i.e. they do not contain valid data; neither before the routine is entered, nor after the routine is exited.

=====

64K CoCo 2 Memory Map
ML Foundation Production

Here's the further modified "ML Foundation Production" Memory Map which I plan to use after this system is fully developed:

```
64K CoCo 2 ML Foundation Production Memory Map
==============================================

Decimal          Address Contents    Hex Address
-------          ----------------    -----------

                 SYSTEM RAM:

0-1023           System Use          0000-03FF

1024-1535        Text Screen Memory  0400-05FF

                 UNUSED:
                 Graphic Screen
1536-3071        Page 1              0600-0BFF

                 BASIC Language
3072-7167        Program Storage     0C00-1BFF

                 Assembly Language
7168-32767       Program Storage     1C00-7FFF

                 SYSTEM UPPER ROM BANK:

                 Assembly Language
32768-65279      Program Storage     8000-FEFF

                 SYSTEM UPPER ROM BANK:

                 Extended
32768-40959      Color BASIC         8000-9FFF

40960-49151      Color BASIC         A000-BFFF

                 Disk Basic or
49152-57343      Cartridge Memory    C000-DFFF
```

57344-65279	Super-Extended (Enhanced) Basic	E000-FEFF
	Hardware Registers, I/O, and Interrupt Vectors:	
65280-65535	Registers & Vectors	FF00-FFFF

____

In production, Graphics Pages, if used, will be allocated within the Assembly Language Program Storage Space in a different, more convenient, location, probably at the high end of Upper RAM.

I will also continue to use the following Low RAM "unused" memory addresses as scratchpad variables wherever needed:

```
$0076 - $0077     2 bytes
$00F3 - $00FF    13 bytes
```

Scratchpad variables are defined as being restricted to within a given isolated routine, i.e. they do not contain valid data; neither before the routine is entered, nor after the routine is exited.

=====

REGXFR: Transfer Variables

So. To begin.

In developing and testing the ML Foundation System, the first thing we need to be able to do is to transfer information between the Assembly Language Routines to be developed and tested, and the BASIC Language Control Programs being used to test those routines.

To provide that capability, I devised a set of Register Variables in memory where BASIC Programs can POKE bytes to be used by the Assembly Language routines, and from which BASIC Programs can PEEK bytes returned from those Routines.

From the Assembly Language side, these memory locations will simply be accessed via suitable LD and ST instructions.

At this point in the system development, since I expect to be routinely jumping between RAM and ROM, I prefer these dedicated addresses to using available unused addresses in Low RAM (i.e. $0000 - $03FF). Despite what available documentation says, I don’t trust the ROM not to mess with such addresses.

We establish these Transfer Variables as follows:

```
00100 *****
00110 *
00120 * REGXFR.ASM
00130 * MDJ 2021/08/02
00140 *
00150 * REGISTER TRANSFER
00160 * VARIABLES
00170 *
00180 * THIS ROUTINE IS USED
00190 * FOR TRANSFERRING
00200 * REGISTER VALUES
00210 * BETWEEN BASIC AND
00220 * ASSEMBLY LANGUAGE
00230 * PROGRAMS
00240 *
00250 * REGA = REGD HIGH BYTE
00260 * REGB = REGD LOW BYTE
00270 *
00280 * THUS REGD = REGA:REGB
00290 * AND REGDH = REGA
00300 *     REGDL = REGB
00310 *
00320 * AS THE 6809 IS A
```

```
                    00330 * BIG-ENDIAN MACHINE
                    00340 * THE LOWER ADDRESS
                    00350 * OF A TWO-BYTE
                    00360 * REGISTER IS THE MOST
                    00370 * SIGNIFICANT BYTE
                    00380 *
                    00390 *****
                    00400
                    00410 * ALTERNATE LABELS
          1C00      00420 REGD    EQU     $1C00
          1C00      00430 REGDH   EQU     $1C00
          1C01      00440 REGDL   EQU     $1C01
          1C02      00450 REGX    EQU     $1C02
          1C04      00460 REGY    EQU     $1C04
          1C06      00470 REGS    EQU     $1C06
          1C08      00480 REGU    EQU     $1C08
          1C0A      00490 REGPC   EQU     $1C0A
                    00500
1C00                00510         ORG     $1C00
                    00520
1C00                00530 REGA    RMB     1
1C01                00540 REGB    RMB     1
1C02                00550 REGXH   RMB     1
1C03                00560 REGXL   RMB     1
1C04                00570 REGYH   RMB     1
1C05                00580 REGYL   RMB     1
1C06                00590 REGSH   RMB     1
1C07                00600 REGSL   RMB     1
1C08                00610 REGUH   RMB     1
1C09                00620 REGUL   RMB     1
1C0A                00630 REGPCH  RMB     1
1C0B                00640 REGPCL  RMB     1
1C0C                00650 REGDP   RMB     1
1C0D                00660 REGCC   RMB     1
                    00670
                    00680 * EXIT
          0000      32767         END

00000 TOTAL ERRORS
```

———-

For example, to transfer a 16-bit value **N1** from a BASIC program to the D Register for use by an Assembly Language routine, and to then get the Assembly Language Routine's result from the X Register, the BASIC program would do something like:

```
100 ‘N1 = 16-BIT NUMBER TO BE
110 ‘TRANSFERRED TO REGISTER D
120 ‘
130 ‘N2 = HIGH BYTE
140 N2 = INT(N1 / 256)
150 ‘
160 ‘N3 = LOW BYTE
170 N3 = INT(N1 - (N2 * 256))
180 ‘
190 ‘REGISTER D TRANSFER
200 ‘VARIABLE ADDRESS =
210 ‘&H1C00:&H1C01
220 POKE &H1C00, N2
230 POKE &H1CO1, N3
240 ‘
250 ‘ GO DO ASSEMBLY LANGUAGE
260 ‘ ROUTINE
270 EXEC &H7000
280 ‘
290 ‘REGISTER X TRANSFER
300 ‘VARIABLE ADDRESS =
310 ‘&H1C02:&H1CO3
320 N2 = PEEK(&H1CO2) ‘HIGH BYTE
330 N3 = PEEK(&H1C03) ‘LOW BYTE
340 ‘
350 ‘ N1 = 16-BIT RESULT
360 N1 = (N2 * 256) + N3
370 PRINT N1
32767 END
```

Meanwhile, the Assembly Language routine would do something like:

```
00100 REGD   EQU       $1C00
00110 REGX   EQU       $1C02
00120
00130                                                                     OR
00140        PSHS      A,B,X
00150
00160 *GET VALUE INTO D
00170 *FROM TRANSFER VARIABLE
00180        LDD       REGD
00190
00200 *DO SOME PROCESSING
00210
00220 *PUT VALUE FROM X
00230 *INTO TRANSFER VARIABLE
00240        STX       REGX
```

```
        00250
        00260 *EXIT
        00270        PULS     A,B,X
        00280        RTS
        00290        END
```

——-

As an actual test of the Transfer Variables, we’ll use the following Assembly Language Routine:

```
                 00100 *****
                 00110 *
                 00120 * TEST0001.ASM
                 00130 * MDJ 2021/08/02
                 00140 *
                 00150 * REGXFR TEST
                 00160 *
                 00170 *****
                 00180
                 00190 * TRANSFER VARIABLES
          1C00   00200 REGA     EQU     $1C00
          1C01   00210 REGB     EQU     $1C01
          1C00   00220 REGD     EQU     $1C00
          1C02   00230 REGX     EQU     $1C02
          1C04   00240 REGY     EQU     $1C04
          1C06   00250 REGS     EQU     $1C06
          1C08   00260 REGU     EQU     $1C08
          1C0A   00270 REGPC    EQU     $1C0A
          1C0C   00280 REGDP    EQU     $1C0C
          1C0D   00290 REGCC    EQU     $1C0D
                 00300
7000             00310          ORG     $7000
                 00320
7000 34   36     00330          PSHS    A,B,X,Y
7002 B6   1C00   00340          LDA     REGA
7005 4C          00350          INCA
7006 4C          00360          INCA
7007 B7   1C00   00370          STA     REGA
700A F6   1C01   00380          LDB     REGB
700D 5C          00390          INCB
700E F7   1C01   00400          STB     REGB
7011 BE   1C02   00410          LDX     REGX
7014 30   05     00420          LEAX    5,X
7016 BF   1C02   00430          STX     REGX
7019 10BE 1C04   00440          LDY     REGY
701D 31   28     00450          LEAY    8,Y
```

```
701F 10BF 1C04     00460          STY     REGY
                   00470
                   00480 * EXIT
7023 35   36       00490          PULS    A,B,X,Y
7025 39            00500          RTS
          0000     32767          END

00000 TOTAL ERRORS
```

along with the following BASIC Language Control Program:

```
1000 '*****
1010 '*
1020 '* TEST0001.BAS
1030 '* MDJ 2021/08/02
1040 '*
1050 '* REGXFR TEST
1060 '*
1070 '*****
1080 '
1090 'SETUP MEMORY
1100 PCLEAR 1
1110 CLEAR 200, &H1C00
1120 '
1130 'LOAD ML ROUTINES
1140 LOADM "REGXFR.BIN"
1150 LOADM "TEST0001.BIN"
1160 '
1170 ' TRANSFER VARIABLES
1180 R0 = &H1C00 'REGA
1190 R1 = &H1C01 'REGB
1200 R2 = &H1C02 'REGXH
1210 R3 = &H1C03 'REGXL
1220 R4 = &H1C04 'REGYH
1230 R5 = &H1C05 'REGYL
1240 R6 = &H1C06 'REGSH
1250 R7 = &H1C07 'REGSL
1260 R8 = &H1C08 'REGUH
1270 R9 = &H1C09 'REGUL
1280 RA = &H1C0A 'REGPCH
1290 RB = &H1C0B 'REGPCL
1300 RC = &H1C0C 'REGDP
1310 RD = &H1C0D 'REGDP
1320 '
1330 'BASIC PREAMBLE
1340 A = 82
1350 B = 27
```

```
1360 X = &H1023 ' 4131 DECIMAL
1370 Y = &HAC37 '44087 DECIMAL
1380 PRINT "ON ENTRY:"
1390 PRINT "  A = "; A
1400 PRINT "  B = "; B
1410 PRINT "  X = "; X
1420 PRINT "  Y = "; Y
1430 POKE R0, A
1440 POKE R1, B
1450 X1 = INT(X/256)
1460 X2 = INT(X-(X1*256))
1470 POKE R2, X1
1480 POKE R3, X2
1490 Y1 = INT(Y/256)
1500 Y2 = INT(Y-(Y1*256))
1510 POKE R4, Y1
1520 POKE R5, Y2
1530 '
1540 'GO DO THE TEST
1550 EXEC &H7000
1560 '
1570 'BASIC POSTAMBLE
1580 A = PEEK(R0)
1590 B = PEEK(R1)
1600 X1 = PEEK(R2)
1610 X2 = PEEK(R3)
1620 X = INT((X1*256)+X2)
1630 Y1 = PEEK(R4)
1640 Y2 = PEEK(R5)
1650 Y = INT((Y1*256)+Y2)
1660 PRINT "ON EXIT:"
1670 PRINT "  A = "; A
1680 PRINT "  B = "; B
1690 PRINT "  X = "; X
1700 PRINT "  Y = "; Y
1710 '
32767 END
```

Results:

```
RUN
ON ENTRY:
  A =  82
  B =  27
  X =  4131
  Y =  44087
ON EXIT:
  A =  84
  B =  28
  X =  4136
  Y =  44095
OK
```

The results are as expected.

=====

VIDCLS: Clear the VIDRAM Screen

Once we’re able to transfer information back-and-forth between BASIC Programs and Assembly Language Routines, the next thing I want to develop is the ability to display information on the screen directly from Assembly Language.

In order to establish that ability, the first step is to be able to clear the screen to make way for the display of the new data. The following routine accomplishes that task.

```
                   00100 *****
                   00110 *
                   00120 * VIDCLS.ASM
                   00130 * MDJ 2021/07/29
                   00140 *
                   00150 * CLEARS THE
                   00160 * VIDEO RAM
                   00170 * AT &H0400 TO &H05FF
                   00180 * TO ALL BYTES = $60
                   00190 * I.E. ALL GREEN BLANKS
                   00200 * I.E. 96 DECIMAL
                   00210 *
                   00220 * ENTRY CONDITIONS:
                   00230 *   NONE
                   00240 *
                   00250 * EXIT CONDITIONS
                   00260 *   NONE
                   00270 *
                   00280 *****
                   00290
                   00300 * FIRST BYTE OF VIDRAM
          0400     00310 VIDRAM  EQU     $0400
                   00320
                   00330 * ONE BYTE PAST THE
                   00340 * LAST BYTE OF VIDRAM
          0600     00350 VIDEND  EQU     $0600
                   00360
                   00370 * GREEN BLANK CHAR CODE
          0060     00380 CHR60   EQU     $60
                   00390
1C0E               00400         ORG     $1C0E
                   00410
1C0E 34   12       00420 VIDCLS  PSHS    A,X
1C10 86   60       00430         LDA     #CHR60
1C12 8E   0400     00440         LDX     #VIDRAM
1C15 A7   80       00450 L1C15   STA     ,X+
```

```
1C17 8C   0600     00460         CMPX    #VIDEND
1C1A 26   F9       00470         BNE     L1C15
                   00480
                   00490 * EXIT
1C1C 35   12       00500         PULS    A,X
1C1E 39            00510         RTS
          0000     32767         END

00000 TOTAL ERRORS
```

Note that we're following the convention of forming all internal labels, i.e labels which will not be accessed from outside of the routine, by using the letter "L" followed by the hexadecimal address where the label is located. This requires some additional work to revise and reassemble the routine, but it also minimizes the potential for conflicting labels between one routine and another.

I also use the same convention for references to scratchpad variables in low memory. I use some of the "unused" addresses in Low RAM ($0000 - $03FF) for these because they're only accessed from within wholly-contained ML routines, i.e. where the BASIC ROM has no opportunity to mess them up.

____

We will use the following Assembly Language Test Routine to test VIDCLS and to then hold the screen as cleared, so we won't miss anything (like something erroneously appearing on the screen and then immediately being obscured by the OK prompt).

```
                   00100 *****
                   00110 *
                   00120 * TEST0002.ASM
                   00130 * MDJ 2021/08/02
                   00140 *
                   00150 * VIDCLS + HOLD TEST
                   00160 *
                   00170 *****
                   00180
                   00190 * EXTERNAL ROUTINE
                   00200 * ADDRESSES
          1C0E     00210 VIDCLS  EQU     $1C0E
                   00220
7000               00230         ORG     $7000
                   00240
7000 34   12       00250         PSHS    A,X
7002 17   AC09     00260         LBSR    VIDCLS
                   00270
                   00280 * HOLD THE SCREEN
7005 86   00       00290 L7005   LDA     #0
```

```
7007 20   FC       00300         BRA     L7005
                   00310
                   00320 * EXIT
7009 35   12       00330         PULS    A,X
700B 39            00340         RTS
          0000     32767         END

00000 TOTAL ERRORS
```

And, we’ll use the following BASIC Language Control Program for the testing:

```
1000 '*****
1010 '*
1020 '* TEST0002.BAS
1030 '* MDJ 2021/08/02
1040 '*
1050 '* VIDCLS + HOLD TEST
1060 '*
1070 '*****
1080 '
1090 'SETUP MEMORY
1100 PCLEAR 1
1110 CLEAR 200, &H1C00
1120 '
1130 'LOAD ML ROUTINES
1140 LOADM "VIDCLS.BIN"
1150 LOADM "REGXFR.BIN"
1160 LOADM "TEST0002.BIN"
1170 '
1180 ' TRANSFER VARIABLES
1190 R0 = &H1C00 'REGA
1200 R1 = &H1C01 'REGB
1210 R2 = &H1C02 'REGXH
1220 R3 = &H1C03 'REGXL
1230 R4 = &H1C04 'REGYH
1240 R5 = &H1C05 'REGYL
1250 R6 = &H1C06 'REGSH
1260 R7 = &H1C07 'REGSL
1270 R8 = &H1C08 'REGUH
1280 R9 = &H1C09 'REGUL
1290 RA = &H1C0A 'REGPCH
1300 RB = &H1C0B 'REGPCL
1310 RC = &H1C0C 'REGDP
1320 RD = &H1C0D 'REGDP
1330 '
1340 'GO DO THE TEST
1350 EXEC &H7000
```

```
1360 '
32767 END
```

Note that I'm continuing to **LOADM** the **REGXFR.BIN** Routine and to specify the Transfer Variable Addresses, even though we're not using the Transfer Variables in this case. As coding and testing continues, I'm also going to continue to include previously tested code; I do this to catch any conflicts which may inadvertently arise. But I'll eventually drop the Transfer Variables list from the BASIC Control Programs for greater clarity.

——

The Result is as expected: a completely blank green screen.

=====

PUTCHR: Put a Character To the VIDRAM Screen At a Specific Position

To continue with the development of our ability to display information on the screen, we have here a little routine that simply puts a character to a specified location in **VIDRAM**. This is perhaps the simplest routine we'll ever have to write during this coding adventure.

PUTCHR does not advance the cursor, and it provides no mechanism for scrolling the screen if necessary. To incorporate those provisions, use **PUTCHA** instead.

```
                    00100 *****
                    00110 *
                    00120 * PUTCHR.ASM
                    00130 * MDJ 2021/08/01
                    00140 *
                    00150 * PUT A CHARACTER CODE
                    00160 * TO THE VIDEO RAM
                    00170 *
                    00180 * ENTRY CONDITIONS:
                    00190 *   A = CHARACTER CODE
                    00200 *   X = SCREEN LOCATION
                    00210 *        ($0400 - $05FF)
                    00220 *
                    00230 * EXIT CONDITIONS:
                    00240 *   NONE
                    00250 *
                    00260 *****
                    00270
1C1F                00280         ORG     $1C1F
                    00290
1C1F A7   84        00300 PUTCHR  STA     ,X
                    00310
                    00320 * EXIT
1C21 39             00330         RTS
          0000      32767         END

00000 TOTAL ERRORS
```

I'll delay testing this routine until the next section.

=====

GETCHR: Get a Charecter From a Specific Position on the VIDRAM Screen

Although a form of input rather than output, the **`GETCHR.ASM`** routine is a mirror of the simple **`PUTCHR.ASM`** routine. It simply retrieves a character code from a specified location in **`VIDRAM`**.

```
                   00100 *****
                   00110 *
                   00120 * GETCHR.ASM
                   00130 * MDJ 2021/08/01
                   00140 *
                   00150 * GET THE CHARACTER CODE
                   00160 * FROM A SPECIFIED
                   00170 * LOCATION
                   00180 * IN VIDEO RAM
                   00190 *
                   00200 * ENTRY CONDITIONS:
                   00210 *   X = SCREEN LOCATION
                   00220 *        ($0400 - $05FF)
                   00230 *
                   00240 * EXIT CONDITIONS:
                   00250 *   A = CHARACTER CODE
                   00260 *
                   00270 *****
                   00280
1CD2               00290         ORG     $1CD2
                   00300
1CD2 A6   84       00310 GETCHR  LDA     ,X
                   00320
                   00330 * EXIT
1CD4 39            00340         RTS
          0000     32767         END

00000 TOTAL ERRORS
```

PUTCHR.ASM and **GETCHR.ASM** are tested together, using the following Assembly Language Test Routine:

```
                         00100 *****
                         00110 *
                         00120 * TEST0003.ASM
                         00130 * MDJ 2021/08/01
                         00140 *
                         00150 * PUTCHR/GETCHR
                         00160 * + HOLD TEST
                         00170 *
                         00180 * CLEARS THE SCREEN
                         00190 * THEN PUTS CHAR CODES
                         00200 * $00 THROUGH $FF
                         00210 * (000-255 DECIMAL)
                         00220 * TO VIDRAM, THEN GETS
                         00230 * THE CHARACTER AT $044D
                         00240 * (=M) AND REPORTS IT
                         00250 * TO THE SCREEN AND THEN
                         00260 * HOLDS THE SCREEN
                         00270 *
                         00280 *****
                         00290
                         00300 * SCREEN ADDRESS
          0400           00310 VIDRAM  EQU     $0400
                         00320
                         00330 * EXTERNAL ROUTINE
                         00340 * ADDRESSES
          1C0E           00350 VIDCLS  EQU     $1C0E
          1C1F           00360 PUTCHR  EQU     $1C1F
          1CD2           00370 GETCHR  EQU     $1CD2
                         00380
7000                     00390         ORG     $7000
                         00400
7000 34   12             00410         PSHS    A,X
                         00420
                         00430 * CLEAR THE SCREEN
7002 17   AC09           00440         LBSR    VIDCLS
                         00450
                         00460 * LOAD THE FIRST
                         00470 * CHAR CODE
7005 86   00             00480         LDA     #$00
                         00490
                         00500 * LOAD THE SCREEN
                         00510 * ADDRESS
7007 8E   0400           00520         LDX     #VIDRAM
                         00530
```

```
                  00540 * FILL THE FIRST PART OF
                  00550 * THE SCREEN WITH THE
                  00560 * CHARACTER SET
700A 17   AC12    00570 L700A   LBSR    PUTCHR
700D 4C           00580         INCA
700E 30   01      00590         LEAX    1,X
7010 81   FF      00600         CMPA    #$FF
7012 26   F6      00610         BNE     L700A
                  00620
                  00630 * POINT TO CHARACTER M'S
                  00640 * ADDRESS IN VIDRAM
7014 8E   044D    00650         LDX     #$044D
                  00660
                  00670 * GET THE CHARACTER M
                  00680 * FROM THE SCREEN
7017 17   ACB8    00690         LBSR    GETCHR
                  00700
                  00710 * POINT FURTHER DOWN
                  00720 * ON THE SCREEN
                  00730 * I.E. IN VIDRAM
701A 8E   0580    00740         LDX     #$0580
                  00750
                  00760 * PUT THE M TO THE
                  00770 * SCREEN
701D 17   ABFF    00780         LBSR    PUTCHR
                  00790
                  00800 * HOLD THE SCREEN
7020 86   00      00810 L7020   LDA     #$00
7022 20   FC      00820         BRA     L7020
                  00830
                  00840 * EXIT
7024 35   12      00850         PULS    A,X
7026 39           00860         RTS
          0000    32767         END

00000 TOTAL ERRORS
```

along with the following BASIC Language Control Program:

```
1000 '*****
1010 '*
1020 '* TEST0003.BAS
1030 '* MDJ 2021/08/02
1040 '*
1050 '* PUTCHR/GETCHR TEST
1060 '*
1070 '*****
```

```
1080 '
1090 'SETUP MEMORY
1100 PCLEAR 1
1110 CLEAR 200, &H1C00
1120 '
1130 'LOAD ML ROUTINES
1140 LOADM "REGXFR.BIN"
1150 LOADM "VIDCLS.BIN"
1160 LOADM "PUTCHR.BIN"
1170 LOADM "GETCHR.BIN"
1180 LOADM "TEST0003.BIN"
1190 '
1200 ' TRANSFER VARIABLES:
1210 '   NOTE THAT REGA IS
1220 '   HIGH BYTE OF REGD
1230 '   SUCH THAT
1240 '   REGD = REGA:REGB
1250 R0 = &H1C00 'REGA
1260 R1 = &H1C01 'REGB
1270 R2 = &H1C02 'REGXH
1280 R3 = &H1C03 'REGXL
1290 R4 = &H1C04 'REGYH
1300 R5 = &H1C05 'REGYL
1310 R6 = &H1C06 'REGSH
1320 R7 = &H1C07 'REGSL
1330 R8 = &H1C08 'REGUH
1340 R9 = &H1C09 'REGUL
1350 RA = &H1C0A 'REGPCH
1360 RB = &H1C0B 'REGPCL
1370 RC = &H1C0C 'REGDP
1380 RD = &H1C0D 'REGDP
1390 '
1400 'GO DO THE TEST
1410 EXEC &H7000
1420 '
32767 END
```

Result:

As expected.

Also note that, except for the reported M at the lower left side of the screen, this is the same result you would get if you ran the following in BASIC:

```
1000 CLS
1010 CD = &H00
1010 FOR AD = &H0400 TO &H04FF
1020 POKE AD, CD
1030 CD = CD + 1
1040 NEXT AD
1050 GOTO 1050
32767 END
```

=====

PUTBYT: Put an 8-bit Number To the VIDRAM Screen As Two Hexadecimal Digits At a Specific Position

Now that we're able to put a character into the **VIDRAM**, its pretty much just a mechanical task to put any text anywhere on the screen. But, it's also important to be able to put numbers to the screen as well.

For the moment, I'm going to concentrate on working with integers only. And, I'm going to restrict myself to only displaying those integers in hexadecimal format. The following **PUTBYT.ASM** Routine places an 8-bit number (byte) on the screen in the form of two consecutive hexadecimal digits, i.e. from "**00**" to "**FF**".

After that, displaying 16-bit, 32-bit, 64-bit integers, etc. is simply a matter of processing one byte after another with this same routine.

PUTBYT does not advance the cursor, and it provides no mechanism for scrolling the screen if necessary. To incorporate those provisions, use **PUTBYA** instead.

```
00100 *****
00110 *
00120 * PUTBYT.ASM
00130 * MDJ 2021/08/27
00140 *
00150 * PUTS AN 8-BIT NUMBER
00160 * TO VIDRAM AS TWO
00170 * HEXADECIMAL DIGITS
00180 *
00190 * ENTRY CONDITIONS:
00200 *   A = THE 8-BIT NUMBER
00210 *   X = SCREEN LOCATION
00220 *        ($0400 - $05FE)
00230 *        CANNOT BE $05FF
00240 *        BECAUSE NEED
00250 *        ROOM TO PUT
00260 *        2 CHARACTERS
00270 *
00280 * EXIT CONDITIONS:
00290 *   X = NEW SCREEN LOC
00300 *        ($0402 - $0600)
00310 *        $0600 INDICATES
```

```
                   00320 *        END OF VIDRAM
                   00330 *        HAS BEEN PASSED
                   00340 *
                   00350 *****
                   00360
                   00370 * SCRATCHPAD VARIABLES
                   00380 * THE 8-BIT NUMBER
          0076     00390 L0076   EQU     $0076
                   00400
                   00410 * THE HIGH NIBBLE
          0077     00420 L0077   EQU     $0077
                   00430
                   00440 * THE LOW NIBBLE
          00F3     00450 L00F3   EQU     $00F3
                   00460
                   00470 * EXTERNAL ROUTINE
                   00480 * ADDRESS
          1C1F     00490 PUTCHR  EQU     $1C1F
                   00500
1CD5               00510         ORG     $1CD5
                   00520
                   00530 * SAVE THE NUMBER
1CD5 97   76       00540 PUTBYT  STA     L0076
                   00550
                   00560 * DIVIDE BY 16
1CD7 44            00570         LSRA
1CD8 44            00580         LSRA
1CD9 44            00590         LSRA
1CDA 44            00600         LSRA
                   00610
                   00620 * SAVE THE HIGH NIBBLE
1CDB 97   77       00630         STA     L0077
                   00640
                   00650 * MULTIPLY BY 16
1CDD 48            00660         LSLA
1CDE 48            00670         LSLA
1CDF 48            00680         LSLA
1CE0 48            00690         LSLA
                   00700
                   00710 * SAVE TEMP RESULT
1CE1 97   F3       00720         STA     L00F3
                   00730
                   00740 * GET THE NUMBER AGAIN
1CE3 96   76       00750         LDA     L0076
                   00760
                   00770 * SUBTRACT TEMP RESULT
1CE5 90   F3       00780         SUBA    L00F3
```

```
                    00790
                    00800 * SAVE LOW NIBBLE
1CE7 97   F3        00810         STA     L00F3
                    00820
                    00830 * IS LOW NIBBLE <= 9
1CE9 81   09        00840         CMPA    #9
                    00850
                    00860 * GO IF NO
1CEB 22   04        00870         BHI     L1CF7
                    00880
                    00890 * ADD ZERO OFFSET
1CED 8B   70        00900         ADDA    #112
1CEF 20   02        00910         BRA     L1CF9
                    00920
                    00930 * ADD "A" OFFSET
1CF1 8B   37        00940 L1CF7   ADDA    #55
                    00950
                    00960 * SAVE LOW NIBBLE CHAR
1CF3 97   F3        00970 L1CF9   STA     L00F3
                    00980
                    00990 * GET HIGH NIBBLE
1CF5 96   77        01000         LDA     L0077
                    01010
                    01020 * IS HIGH NIBBLE <= 9
1CF7 81   09        01030         CMPA    #9
                    01040
                    01050 * GO IF NO
1CF9 22   04        01060         BHI     L1D07
                    01070
                    01080 * ADD ZERO OFFSET
1CFB 8B   70        01090         ADDA    #112
1CFD 20   02        01100         BRA     L1D09
                    01110
                    01120 * ADD "A" OFFSET
1CFF 8B   37        01130 L1D07   ADDA    #55
                    01140
                    01150 * PUT HIGH NIBBLE CHAR
                    01160 * TO VIDRAM
1D01 17   FF1B      01170 L1D09   LBSR    PUTCHR
                    01180
                    01190 * INCREMENT VIDRAM PTR
1D04 30   01        01200         LEAX    1,X
                    01210
                    01220 * GET LOW NIBBLE CHAR
1D06 96   F3        01230         LDA     L00F3
                    01240
                    01250 * PUT LOW NIBBLE CHAR
```

```
                    01260 * TO VIDRAM
1D08 17   FF14      01270       LBSR    PUTCHR
                    01280
                    01290 * INCREMENT VIDRAM PTR
1D0B 30   01        01300       LEAX    1,X
                    01310
                    01320 * EXIT
1D0D 39             01330       RTS
          0000      32767       END

00000 TOTAL ERRORS
```

——-

The following Assembly Language Test Routine will be used:

```
                    00100 *****
                    00110 *
                    00120 * TEST0004.ASM
                    00130 * MDJ 2021/08/27
                    00140 *
                    00150 * PUTBYT + HOLD TEST
                    00160 *
                    00170 * PUTS ALL BYTES
                    00180 * ($00 - $FF) TO THE
                    00190 * SCREEN, WITHOUT ANY
                    00200 * SPACES, AND THEN
                    00210 * HOLDS THE SCREEN
                    00220 *
                    00230 *****
                    00240
                    00250 * SCREEN ADDRESSES
          0400      00260 VIDRAM  EQU     $0400
          0600      00270 VIDEND  EQU     $0600
                    00280
                    00290 * EXTERNAL ROUTINE
                    00300 * ADDRESSES
          1C0E      00310 VIDCLS  EQU     $1C0E
          1CD5      00320 PUTBYT  EQU     $1CD5
                    00330
7000                00340         ORG     $7000
                    00350
7000 34   16        00360         PSHS    A,B,X
                    00370
                    00380 * CLEAR THE SCREEN
7002 17   AC09      00390         LBSR    VIDCLS
                    00400
```

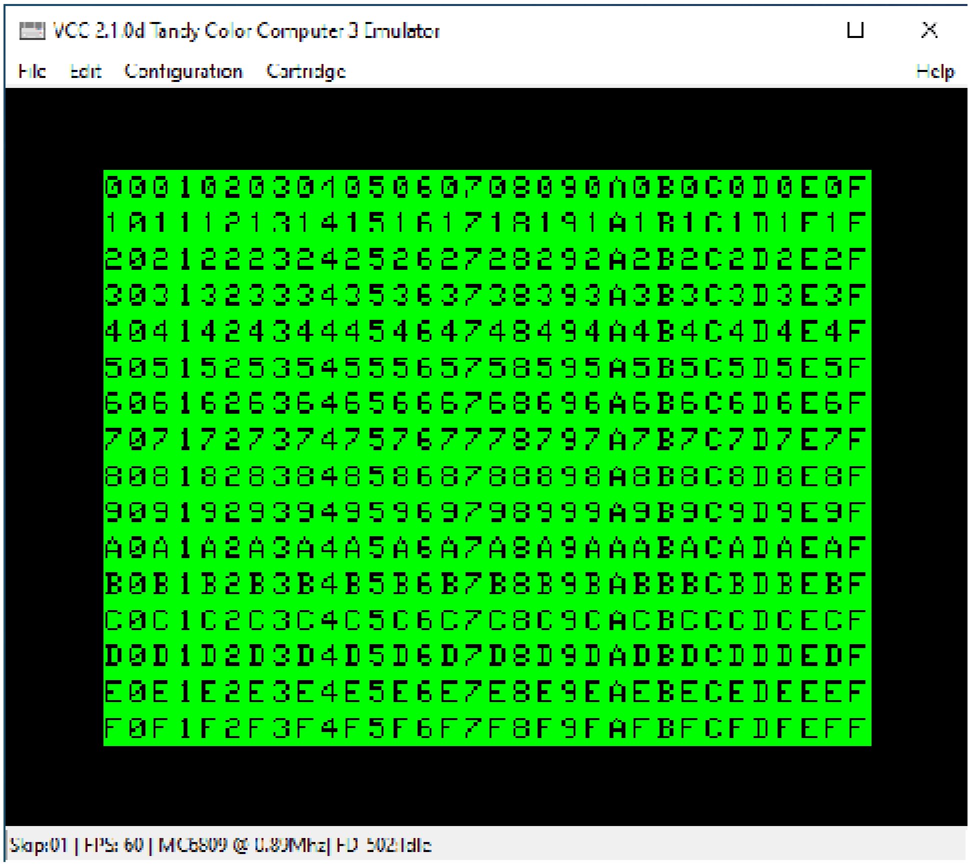
```
                 00410 * LOAD THE FIRST
                 00420 * BYTE VALUE
7005 86   00     00430        LDA     #$00
                 00440
                 00450 * LOAD THE SCREEN
                 00460 * ADDRESS
7007 8E   0400   00470        LDX     #VIDRAM
                 00480
                 00490 * SAVE THE BYTE VALUE
700A 1F   89     00500        TFR     A,B
                 00510
                 00520 * GO PUT BYTE TO SCREEN
700C 17   ACC6   00530 L700C  LBSR    PUTBYT
                 00540
                 00550 * ARE WE DONE?
700F 8C   0600   00560        CMPX    #VIDEND
                 00570
                 00580 * GO IF YES
7012 24   05     00590        BHS     L7019
                 00600
                 00610 * GET NEXT BYTE VALUE
7014 5C          00620        INCB
7015 1F   98     00630        TFR     B,A
7017 20   F3     00640        BRA     L700C
                 00650
                 00660 * HOLD THE SCREEN
7019 86   00     00670 L7019  LDA     #$00
701B 20   FC     00680        BRA     L7019
                 00690
                 00700 * EXIT
701D 35   16     00710        PULS    A,B,X
701F 39          00720        RTS
          0000   32767        END

00000 TOTAL ERRORS
```

along with the following BASIC Language Control Program:

```
1000 '*****
1010 '*
1020 '* TEST0004.BAS
1030 '* MDJ 2021/08/27
1040 '*
1050 '* PUTBYT + HOLD TEST
1060 '*
1070 '*****
1080 '
```

```
1090 'SETUP MEMORY
1100 PCLEAR 1
1110 CLEAR 200, &H1C00
1120 '
1130 'LOAD ML ROUTINES
1140 LOADM "REGXFR.BIN"
1150 LOADM "VIDCLS.BIN"
1160 LOADM "PUTCHR.BIN"
1170 LOADM "GETCHR.BIN"
1180 LOADM "PUTBYT.BIN"
1190 LOADM "TEST0004.BIN"
1200 '
1210 ' TRANSFER VARIABLES:
1220 '   NOTE THAT REGA IS
1230 '   HIGH BYTE OF REGD
1240 '   SUCH THAT
1250 '   REGD = REGA:REGB
1260 R0 = &H1C00 'REGA
1270 R1 = &H1C01 'REGB
1280 R2 = &H1C02 'REGXH
1290 R3 = &H1C03 'REGXL
1300 R4 = &H1C04 'REGYH
1310 R5 = &H1C05 'REGYL
1320 R6 = &H1C06 'REGSH
1330 R7 = &H1C07 'REGSL
1340 R8 = &H1C08 'REGUH
1350 R9 = &H1C09 'REGUL
1360 RA = &H1C0A 'REGPCH
1370 RB = &H1C0B 'REGPCL
1380 RC = &H1C0C 'REGDP
1390 RD = &H1C0D 'REGDP
1400 '
1410 'GO DO THE TEST
1420 EXEC &H7000
1430 '
32767 END
```

Result:



As expected.

=====

SCROLL: Scroll the VIDRAM Screen WITH A TESTING ERROR

I expect that much of what I'm planning to do with the ML Foundation will involve putting information to specific locations in **`VIDRAM`**, rather than a continuous scrolling output.

Nonetheless, there may be times and applications where the ability to scroll the screen will be useful. And this seems like a convenient spot to cover scrolling. So…

```
                         00100 *****
                         00110 *
                         00120 * SCROLL.ASM
                         00130 * MDJ 2021/08/27
                         00140 *
                         00150 * SCROLLS THE SCREEN
                         00160 *
                         00170 * ENTRY CONDITIONS
                         00180 *   NONE
                         00190 *
                         00200 * EXIT CONDITIONS
                         00210 *   NONE
                         00220 *
                         00230 *****
                         00240
                         00250 * SCREEN ADDRESSES
          0400           00260 VIDRAM  EQU     $0400
          0600           00270 VIDEND  EQU     $0600
          0420           00280 VIDL01  EQU     $0420
                         00290
1D17                     00300         ORG     $1D17
                         00310
1D17 34   32             00320 SCROLL  PSHS    A,X,Y
                         00330
                         00340 * Y = SOURCE POINTER
                         00350 * X = TARGET POINTER
                         00360
                         00370 * POINT X TO FIRST LINE
1D19 8E   0400           00380         LDX     #VIDRAM
                         00390
                         00400 * POINT Y TO SECOND LINE
1D1C 108E 0420           00410         LDY     #VIDL01
                         00420
                         00430 * SCROLL THE SCREEN
1D20 A6   A0             00440 L1D20   LDA     ,Y+
1D22 A7   80             00450         STA     ,X+
```

```
                    00460
                    00470 * ARE WE DONE?
1D24 108C 0600      00480        CMPY    #VIDEND
                    00490
                    00500 * GO IF NO
1D28 25   F6        00510        BLO     L1D20
                    00520
                    00530 * CLEAR THE LAST LINE
                    00540 * LOAD BLANK GREEN CHAR
1D2A 86   60        00550        LDA     #96
1D2C A7   80        00560 L1D2C  STA     ,X+
                    00570
                    00580 * ARE WE DONE?
1D2E 8C   0600      00590        CMPX    #VIDEND
                    00600
                    00610 * GO IF NO
1D31 25   F9        00620        BLO     L1D2C
                    00630
                    00640 * EXIT
1D33 35   32        00650        PULS    A,X,Y
1D35 39             00660        RTS
          0000      32767        END

00000 TOTAL ERRORS
```

-——-

Testing the scrolling will just involve repeating the **TEST0004** procedure from the **PUTBYT** Chapter, but with the addition of a one-line scroll after the screen is filled. The Assembly Language Test Routine will be:

```
00100 *****
00110 *
00120 * TEST0005.ASM
00130 * MDJ 2021/08/27
00140 *
00150 * SCROLL TEST
00160 *
00170 * PUTS ALL BYTES
00180 * ($00 - $FF) TO THE
00190 * SCREEN, WITHOUT ANY
00200 * SPACES, THEN SCROLLS
00210 * THE SCREEN ONE LINE,
00220 * AND THEN HOLDS THE
00230 * SCREEN
00240 *
00250 *****
```

```
                    00260
                    00270 * SCREEN ADDRESSES
          0400      00280 VIDRAM  EQU     $0400
          0600      00290 VIDEND  EQU     $0600
                    00300
                    00310 * EXTERNAL ROUTINE
                    00320 * ADDRESSES
          1C0E      00330 VIDCLS  EQU     $1C0E
          1CD5      00340 PUTBYT  EQU     $1CD5
          1D17      00350 SCROLL  EQU     $1D17
                    00360
7000                00370         ORG     $7000
                    00380
7000 34   16        00390         PSHS    A,B,X
                    00400
                    00410 * CLEAR THE SCREEN
7002 17   AC09      00420         LBSR    VIDCLS
                    00430
                    00440 * LOAD THE FIRST
                    00450 * BYTE VALUE
7005 86   00        00460         LDA     #$00
                    00470
                    00480 * LOAD THE SCREEN
                    00490 * ADDRESS
7007 8E   0400      00500         LDX     #VIDRAM
                    00510
                    00520 * SAVE THE BYTE VALUE
700A 1F   89        00530         TFR     A,B
                    00540
                    00550 * GO PUT BYTE TO SCREEN
700C 17   ACC6      00560 L700C   LBSR    PUTBYT
                    00570
                    00580 * ARE WE DONE?
700F 8C   0600      00590         CMPX    #VIDEND
                    00600
                    00610 * GO IF YES
7012 24   08        00620         BHS     L701C
                    00630
                    00640 * GET NEXT BYTE VALUE
7014 5C             00650         INCB
7015 1F   98        00660         TFR     B,A
7017 20   F3        00670         BRA     L700C
                    00680
                    00690 * SCROLL THE SCREEN
                    00700 * ONE LINE
7019 17   ACFB      00710         LBSR    SCROLL
                    00720
```

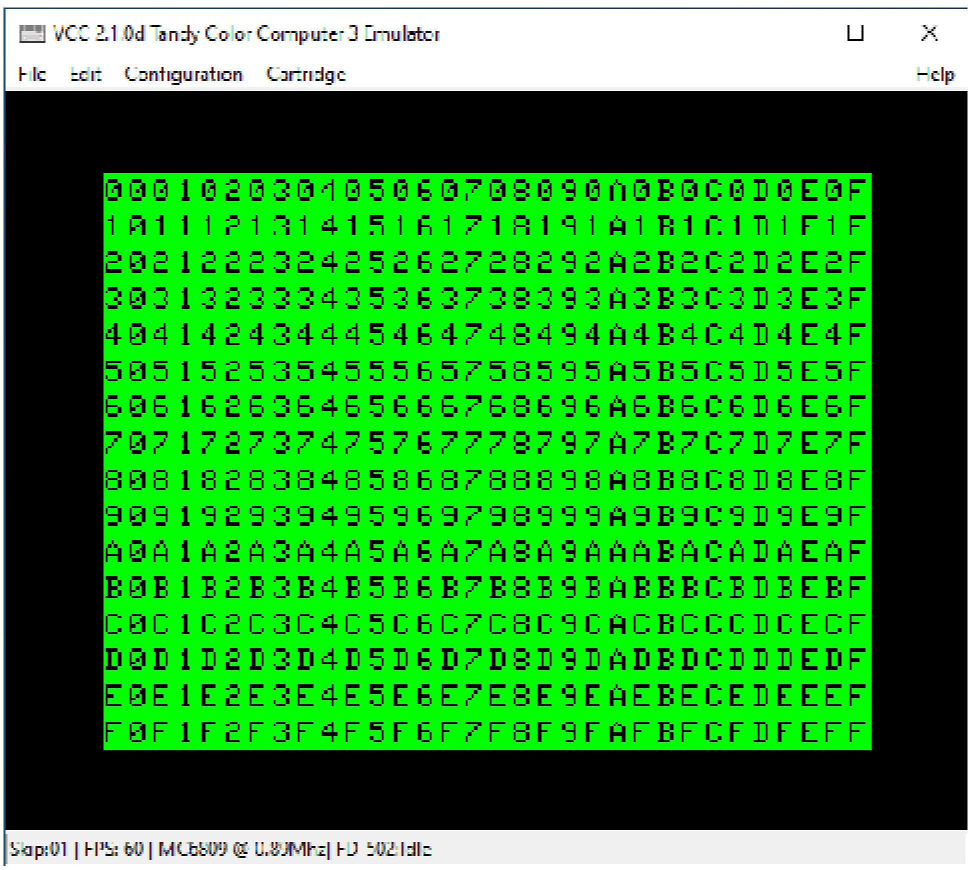
```
                    00730 * HOLD THE SCREEN
701C 86   00        00740 L701C   LDA     #$00
701E 20   FC        00750         BRA     L701C
                    00760
                    00770 * EXIT
7020 35   16        00780         PULS    A,B,X
7022 39             00790         RTS
          0000      32767         END

00000 TOTAL ERRORS
```

And the BASIC Language Control Program is:

```
1000 '*****
1010 '*
1020 '* TEST0005.BAS
1030 '* MDJ 2021/08/27
1040 '*
1050 '* SCROLL TEST
1060 '*
1070 '*****
1080 '
1090 'SETUP MEMORY
1100 PCLEAR 1
1110 CLEAR 200, &H1C00
1120 '
1130 'LOAD ML ROUTINES
1140 LOADM "REGXFR.BIN"
1150 LOADM "VIDCLS.BIN"
1160 LOADM "PUTCHR.BIN"
1170 LOADM "GETCHR.BIN"
1180 LOADM "PUTBYT.BIN"
1190 LOADM "SCROLL.BIN"
1200 LOADM "TEST0005.BIN"
1210 '
1220 ' TRANSFER VARIABLES:
1230 '   NOTE THAT REGA IS
1240 '   HIGH BYTE OF REGD
1250 '   SUCH THAT
1260 '   REGD = REGA:REGB
1270 R0 = &H1C00 'REGA
1280 R1 = &H1C01 'REGB
1290 R2 = &H1C02 'REGXH
1300 R3 = &H1C03 'REGXL
1310 R4 = &H1C04 'REGYH
1320 R5 = &H1C05 'REGYL
1330 R6 = &H1C06 'REGSH
```

```
1340 R7 = &H1C07 'REGSL
1350 R8 = &H1C08 'REGUH
1360 R9 = &H1C09 'REGUL
1370 RA = &H1C0A 'REGPCH
1380 RB = &H1C0B 'REGPCL
1390 RC = &H1C0C 'REGDP
1400 RD = &H1C0D 'REGDP
1410 '
1420 'GO DO THE TEST
1430 EXEC &H7000
1440 '
32767 END
```

Result:



*****

This is NOT what was expected !

This is the same as the **PUTBYT** result itself: The screen did not scroll.

Let’s investigate.

We do so in the next Section.

=====

SCROLL: Scroll the VIDRAM Screen
CHECKING THE TESTING ERROR

The screen did not scroll. A code walk-through of **SCROLL.ASM** indicates that everything initially seems okay. Are we actually getting to the **SCROLL** Routine? Let's check.

Let's add some error checking to **TEST0005.ASM**.

First, we add the address of PUTCHR at Line 351.

Second, we add an Error Check in lines 701-707. This will check to see if **TEST0005** is actually getting to the **SCROLL** at Line 710. If yes, but the scroll does not occur, then the letter "A" should appear in the first character position in the last line on the screen. If yes, and the scroll DOES occur, then the letter "A" should appear in the first character position in the next-to-last line on the screen (i.e. the last line which isn't blank).

Third, we add another Check in lines 711-717 to check if we get back from the **SCROLL** Routine. If yes, but the scroll did not occur, then the letter "B" should appear in the second character position in the next-to-last line on the screen. If yes, and the scroll DOES occur, then the letter "B" should still appear in the second character position in the next-to-last line on the screen (i.e. the last line which isn't blank).

```
                 00100 *****
                 00110 *
                 00120 * TEST005.ASM
                 00130 * MDJ 2021/08/27
                 00140 *
                 00150 * SCROLL TEST
                 00160 *
                 00170 * PUTS ALL BYTES
                 00180 * ($00 - $FF) TO THE
                 00190 * SCREEN, WITHOUT ANY
                 00200 * SPACES, THEN SCROLLS
                 00210 * THE SCREEN ONE LINE,
                 00220 * AND THEN HOLDS THE
                 00230 * SCREEN
                 00240 *
                 00250 *****
                 00260
                 00270 * SCREEN ADDRESSES
0400             00280 VIDRAM  EQU     $0400
0600             00290 VIDEND  EQU     $0600
                 00300
                 00310 * EXTERNAL ROUTINE
```

```
                   00320 * ADDRESSES
          1C0E     00330 VIDCLS  EQU     $1C0E
          1CD5     00340 PUTBYT  EQU     $1CD5
          1D17     00350 SCROLL  EQU     $1D17
          1C1F     00351 PUTCHR  EQU     $1C1F
                   00360
7000               00370         ORG     $7000
                   00380
7000 34   16       00390         PSHS    A,B,X
                   00400
                   00410 * CLEAR THE SCREEN
7002 17   AC09     00420         LBSR    VIDCLS
                   00430
                   00440 * LOAD THE FIRST
                   00450 * BYTE VALUE
7005 86   00       00460         LDA     #$00
                   00470
                   00480 * LOAD THE SCREEN
                   00490 * ADDRESS
7007 8E   0400     00500         LDX     #VIDRAM
                   00510
                   00520 * SAVE THE BYTE VALUE
700A 1F   89       00530         TFR     A,B
                   00540
                   00550 * GO PUT BYTE TO SCREEN
700C 17   ACC6     00560 L700C   LBSR    PUTBYT
                   00570
                   00580 * ARE WE DONE?
700F 8C   0600     00590         CMPX    #VIDEND
                   00600
                   00610 * GO IF YES
7012 24   20       00620         BHS     L701C
                   00630
                   00640 * GET NEXT BYTE VALUE
7014 5C            00650         INCB
7015 1F   98       00660         TFR     B,A
7017 20   F3       00670         BRA     L700C
                   00680
                   00690 * SCROLL THE SCREEN
                   00700 * ONE LINE
                   00701 *** ERRCHK: GOING
7019 34   12       00702         PSHS    A,X
701B 86   41       00703         LDA     #65
701D 8E   05E0     00704         LDX     #$05E0
7020 17   ABFC     00705         LBSR    PUTCHR
7023 35   12       00706         PULS    A,X
                   00707 *** END ERRCHK
```

```
7025 17   ACEF     00710          LBSR    SCROLL
                   00711 *** ERRCHK: COMING
7028 34   12       00712          PSHS    A,X
702A 86   42       00713          LDA     #66
702C 8E   05E1     00714          LDX     #$05C1
702F 17   ABED     00715          LBSR    PUTCHR
7032 35   12       00716          PULS    A,X
                   00717 *** END ERRCHK
                   00720
                   00730 * HOLD THE SCREEN
7034 86   00       00740 L701C    LDA     #$00
7036 20   FC       00750          BRA     L701C
                   00760
                   00770 * EXIT
7038 35   16       00780          PULS    A,B,X
703A 39            00790          RTS
          0000     32767          END

00000 TOTAL ERRORS
```

We run the Control Program with this new Test Routine …

——-

The result is the same as before: No scrolling and no “AB” at the beginning of the last or next-to-last line.

Bah !!

We’re not getting to the **`SCROLL`** at all!

I went over the Test Routine more carefully.

Aha !

When the Test is done putting all the 8-bit numbers to the screen, Line 620 branches to Label **`L701C`** (Line 740); the same place it did before. It’s branching right around the call to the **`SCROLL`** Routine we just added.

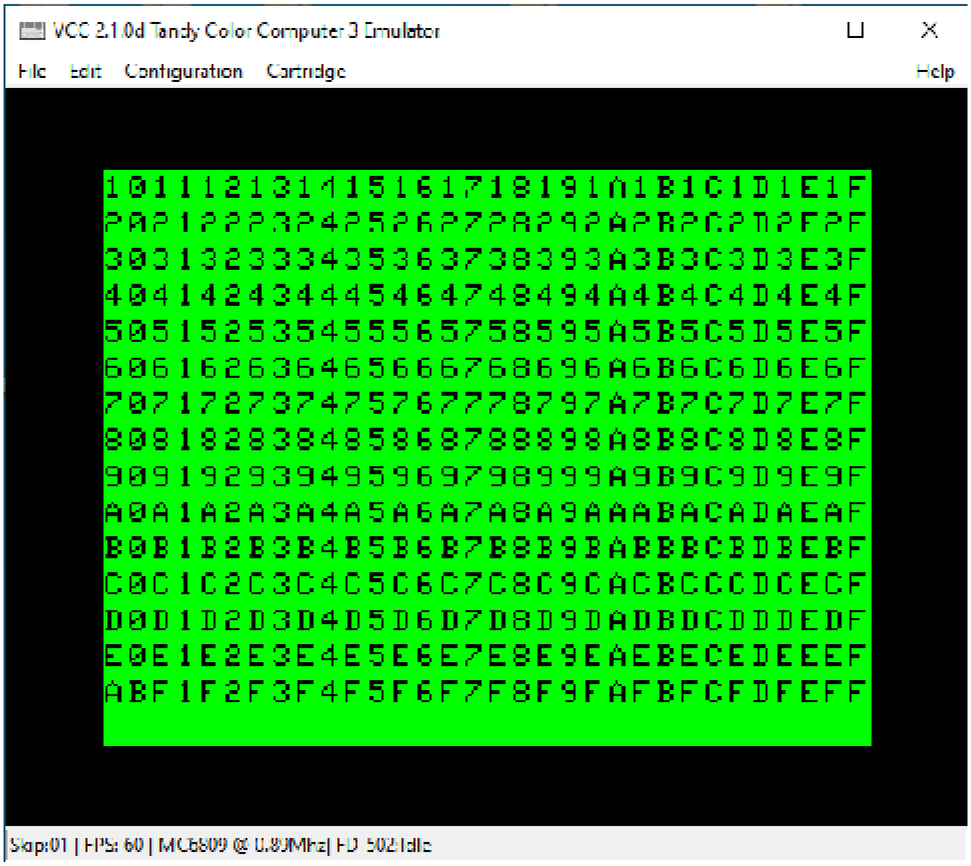
> (Please wait for a moment while I smack myself in the forehead and utter the words “BIG DUMMY!”)

So, I added a new temporary Label **`L0001`** to Line 702 and changed Line 620 to branch to that Label instead:

```
                    00100 *****
                    00110 *
                    00120 * TEST0005.ASM
                    00130 * MDJ 2021/08/27
                    00140 *
                    00150 * SCROLL TEST
                    00160 *
                    00170 * PUTS ALL BYTES
                    00180 * ($00 - $FF) TO THE
                    00190 * SCREEN, WITHOUT ANY
                    00200 * SPACES, THEN SCROLLS
                    00210 * THE SCREEN ONE LINE,
                    00220 * AND THEN HOLDS THE
                    00230 * SCREEN
                    00240 *
                    00250 *****
                    00260
                    00270 * SCREEN ADDRESSES
          0400      00280 VIDRAM  EQU     $0400
          0600      00290 VIDEND  EQU     $0600
                    00300
                    00310 * EXTERNAL ROUTINE
                    00320 * ADDRESSES
          1C0E      00330 VIDCLS  EQU     $1C0E
          1CD5      00340 PUTBYT  EQU     $1CD5
          1D17      00350 SCROLL  EQU     $1D17
          1C1F      00351 PUTCHR  EQU     $1C1F
                    00360
7000                00370         ORG     $7000
                    00380
7000 34   16        00390         PSHS    A,B,X
                    00400
                    00410 * CLEAR THE SCREEN
7002 17   AC09      00420         LBSR    VIDCLS
                    00430
                    00440 * LOAD THE FIRST
                    00450 * BYTE VALUE
7005 86   00        00460         LDA     #$00
                    00470
                    00480 * LOAD THE SCREEN
                    00490 * ADDRESS
7007 8E   0400      00500         LDX     #VIDRAM
                    00510
                    00520 * SAVE THE BYTE VALUE
```

```
700A 1F   89       00530         TFR     A,B
                   00540
                   00550 * GO PUT BYTE TO SCREEN
700C 17   ACC6     00560 L700C   LBSR    PUTBYT
                   00570
                   00580 * ARE WE DONE?
700F 8C   0600     00590         CMPX    #VIDEND
                   00600
                   00610 * GO IF YES
7012 24   05       00620         BHS     L0001
                   00630
                   00640 * GET NEXT BYTE VALUE
7014 5C            00650         INCB
7015 1F   98       00660         TFR     B,A
7017 20   F3       00670         BRA     L700C
                   00680
                   00690 * SCROLL THE SCREEN
                   00700 * ONE LINE
                   00701 *** ERRCHK: GOING
7019 34   12       00702 L0001   PSHS    A,X
701B 86   41       00703         LDA     #65
701D 8E   05E0     00704         LDX     #$05E0
7020 17   ABFC     00705         LBSR    PUTCHR
7023 35   12       00706         PULS    A,X
                   00707 *** END ERRCHK
7025 17   ACEF     00710         LBSR    SCROLL
                   00711 *** ERRCHK: COMING
7028 34   12       00712         PSHS    A,X
702A 86   42       00713         LDA     #66
702C 8E   05E1     00714         LDX     #$05E1
702F 17   ABED     00715         LBSR    PUTCHR
7032 35   12       00716         PULS    A,X
                   00717 *** END ERRCHK
                   00720
                   00730 * HOLD THE SCREEN
7034 86   00       00740 L701C   LDA     #$00
7036 20   FC       00750         BRA     L701C
                   00760
                   00770 * EXIT
7038 35   16       00780         PULS    A,B,X
703A 39            00790         RTS
          0000     32767         END

00000 TOTAL ERRORS
```

Result:



Now, THIS is what we expected.

The screen has scrolled up one line and the “AB” is also where we expected; at the beginning of the next-to-last line on the screen. (i.e. the last line which isn’t blank).

=====

SCROLL: Scroll the VIDRAM Screen CORRECTED

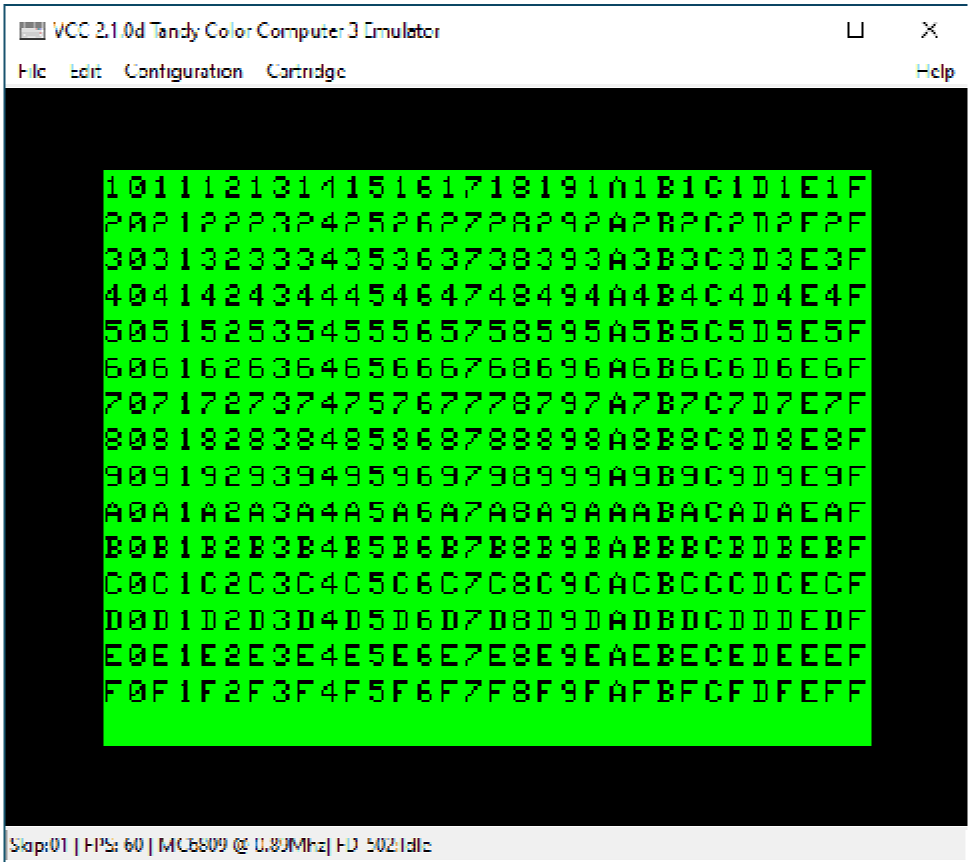
So I removed the error checking code and added a new Label to Line 710 for Line 620 to branch to:

```
                   00100 *****
                   00110 *
                   00120 * TEST0005.ASM
                   00130 * MDJ 2021/08/27
                   00140 *
                   00150 * SCROLL TEST
                   00160 *
                   00170 * PUTS ALL BYTES
                   00180 * ($00 - $FF) TO THE
                   00190 * SCREEN, WITHOUT ANY
                   00200 * SPACES, THEN SCROLLS
                   00210 * THE SCREEN ONE LINE,
                   00220 * AND THEN HOLDS THE
                   00230 * SCREEN
                   00240 *
                   00250 *****
                   00260
                   00270 * SCREEN ADDRESSES
          0400     00280 VIDRAM  EQU     $0400
          0600     00290 VIDEND  EQU     $0600
                   00300
                   00310 * EXTERNAL ROUTINE
                   00320 * ADDRESSES
          1C0E     00330 VIDCLS  EQU     $1C0E
          1CD5     00340 PUTBYT  EQU     $1CD5
          1D17     00350 SCROLL  EQU     $1D17
                   00360
7000               00370         ORG     $7000
                   00380
7000 34   16       00390         PSHS    A,B,X
                   00400
                   00410 * CLEAR THE SCREEN
7002 17   AC09     00420         LBSR    VIDCLS
                   00430
                   00440 * LOAD THE FIRST
                   00450 * BYTE VALUE
7005 86   00       00460         LDA     #$00
                   00470
                   00480 * LOAD THE SCREEN
```

```
                        00490 * ADDRESS
7007 8E   0400          00500         LDX     #VIDRAM
                        00510
                        00520 * SAVE THE BYTE VALUE
700A 1F   89            00530         TFR     A,B
                        00540
                        00550 * GO PUT BYTE TO SCREEN
700C 17   ACC6          00560 L700C   LBSR    PUTBYT
                        00570
                        00580 * ARE WE DONE?
700F 8C   0600          00590         CMPX    #VIDEND
                        00600
                        00610 * GO IF YES
7012 24   05            00620         BHS     L7019
                        00630
                        00640 * GET NEXT BYTE VALUE
7014 5C                 00650         INCB
7015 1F   98            00660         TFR     B,A
7017 20   F3            00670         BRA     L700C
                        00680
                        00690 * SCROLL THE SCREEN
                        00700 * ONE LINE
7019 17   ACFB          00710 L7019   LBSR    SCROLL
                        00720
                        00730 * HOLD THE SCREEN
701C 86   00            00740 L701C   LDA     #$00
701E 20   FC            00750         BRA     L701C
                        00760
                        00770 * EXIT
7020 35   16            00780         PULS    A,B,X
7022 39                 00790         RTS
          0000          32767         END

00000 TOTAL ERRORS
```

We again run the Control Program with this new Test Routine …

Result:



As expected.

(Finally! Whew!)

=====

PUTCHA: Put a Character To the VIDRAM Screen at the Cursor Position and Advance the Cursor

This routine is designed to function with continuous scrolling output.

PUTCHA advances the cursor, and it also scrolls the screen when required. If those provisions are not necessary for a particular application, use **PUTCHR** instead: it uses less resources.

```
        *****
        *
        * PUTCHA.ASM
        * MDJ 2021/08/29
        *
        * PUT A CHARACTER CODE
        * TO THE VIDEO RAM
        * AT THE CURSOR POSITION
        * AND ADVANCE THE CURSOR
        *
        * SCROLLS THE SCREEN
        * IF REQUIRED
        *
        * ENTRY CONDITIONS:
        *   A = CHARACTER CODE
        *
        * EXIT CONDITIONS:
        *   NONE
        *
        *****

        * LOW RAM CURSOR ADDRESS
0088    CURPOS  EQU     $0088

        * START OF THE LAST
        * LINE OF VIDRAM
05E0    VIDL15  EQU     $05E0

        * ONE BYTE PAST THE
        * END OF VIDRAM
0600    VIDEND  EQU     $0600

```

```
                 00420 * EXTERNAL ROUTINE
                 00430 * ADDRESS
          1D17   00440 SCROLL  EQU     $1D17
                 00450
1D36             00460         ORG     $1D36
                 00470
1D36 34   10     00480 PUTCHA  PSHS    X
                 00490
                 00500 * GET THE CURSOR
1D38 9E   88     00510         LDX     CURPOS
                 00520
                 00530 * IS PRE-SCROLL REQUIRED?
1D3A 8C   0600   00540         CMPX    #VIDEND
                 00550
                 00560 * GO IF NO
1D3D 25   06     00570         BLO     L1D45
                 00580
                 00590 * SCROLL ONE LINE
1D3F 17   FFD5   00600         LBSR    SCROLL
1D42 8E   05E0   00610         LDX     #VIDL15
                 00620
                 00630 * PUT THE CHARACTER CODE
1D45 A7   80     00640 L1D45   STA     ,X+
                 00650
                 00660 * END OF VIDRAM?
1D47 8C   0600   00670         CMPX    #VIDEND
                 00680
                 00690 * GO IF NO
1D4A 25   06     00700         BLO     L1D52
                 00710
                 00720 * SCROLL ONE LINE
1D4C 17   FFC8   00730         LBSR    SCROLL
1D4F 8E   05E0   00740         LDX     #VIDL1
                 00750
                 00760 * STORE NEW CURSOR
1D52 9F   88     00770 L1D52   STX     CURPOS
                 00780
                 00790 * EXIT
1D54 35   10     00800         PULS    X
1D56 39          00810         RTS
          0000   32767         END

00000 TOTAL ERRORS
```

Note that this routine depends upon the Direct Page (DP) Register remaining at $00.

The Assembly Language Test Routine:

```
                    00100 *****
                    00110 *
                    00120 * TEST0006.ASM
                    00130 * MDJ 2021/08/29
                    00140 *
                    00150 * PUTCHA TEST
                    00160 *
                    00170 *****
                    00180
                    00190 * EXTERNAL ROUTINE
                    00200 * ADDRESS
          1D36      00210 PUTCHA  EQU     $1D36
                    00220
7000                00230         ORG     $7000
                    00240
7000 34   20        00250         PSHS    Y
7002 20   0D        00260         BRA     L7011
                    00270
7004      4A        00280 STRING  FCC     /JESUS/
          45
          53
          55
          53
7009      60        00281         FCB     $60
700A      4C        00282         FCC     /LIVES/
          49
          56
          45
          53
700F      61        00283         FCB     $61
7010      00        00290         FCB     $00
                    00300
                    00310 * POINT TO THE STRING
7011 108E 7004      00320 L7011   LDY     #STRING
                    00330
                    00340 * GET A CHARACTER
7015 A6   A0        00350 L7015   LDA     ,Y+
                    00360
                    00370 * GO IF NULL TERMINATOR
7017 27   05        00380         BEQ     L701E
                    00390
                    00400 * PUT IT TO VIDRAM
7019 17   AD1A      00410         LBSR    PUTCHA
701C 20   F7        00420         BRA     L7015
```

```
                    00430
                    00440 * EXIT
701E 35   20        00450 L701E   PULS    Y
7020 39             00460         RTS
          0000      32767         END

00000 TOTAL ERRORS
```

Note that Lines 280-290 were originally:

```
7004      4A        00280 STRING  FCC     /JESUS LIVES!/
          45
          53
          55
          53
          20
          4C
          49
          56
          45
          53
          21
7010      00        00290         FCB     $00
```

The additions and rearrangements were necessary because, in Machine Language, were doing essentially the same thing as the **BASIC POKE** mechanism here. We're inserting the character codes directly into **VIDRAM** memory rather than using the **PRINT** mechanism to display them.

The **PRINT** mechanism codes for a space ($20 = 032 decimal) and an exclamation point ($21 = 033 decimal) display as Reversed (i.e. green-on-black) instead of the Standard black-on-green when placed on the screen by the **POKE** mechanism instead (cf. MDJ01).

Thus, we have to substitute the **POKE** mechanism codes for the space ($60 = 096 decimal) and the exclamation point ($61 = 097 decimal) instead.

Later, we'll handle this sort of thing through the **PK2PRT** and **PRT2PK** Translation Routines.

——

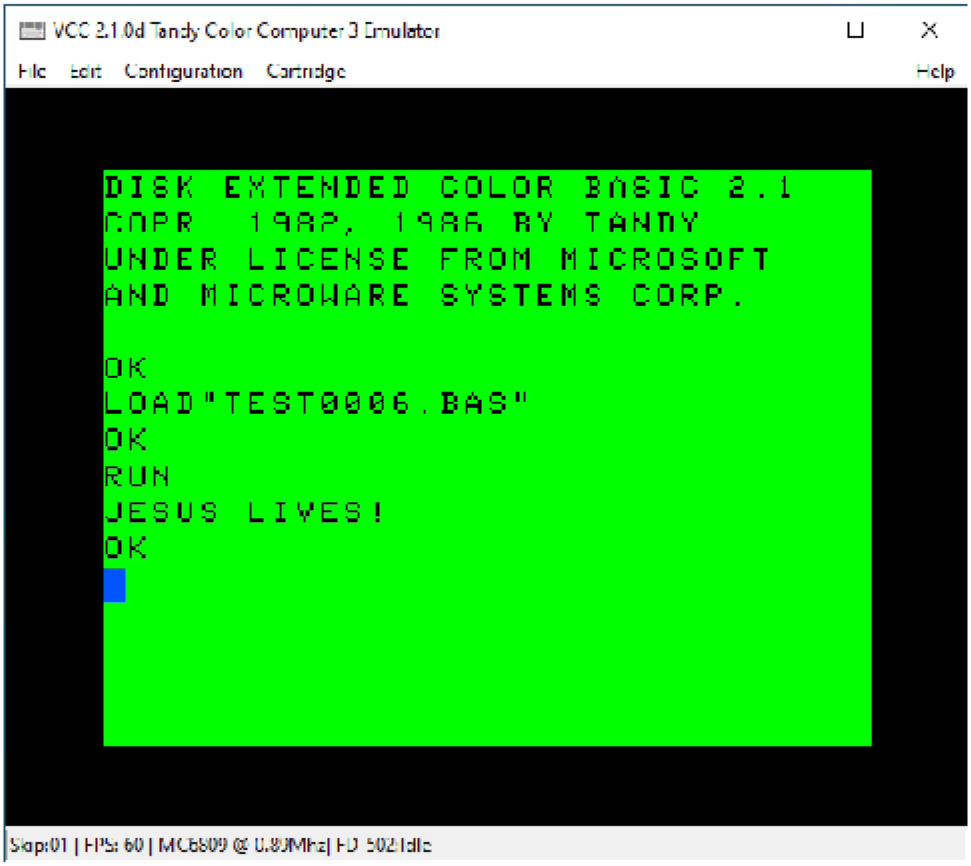
The BASIC Language Control Program

```
1000 '*****
1010 '*
1020 '* TEST0006.BAS
1030 '* MDJ 2021/08/29
1040 '*
1050 '* PUTCHA TEST
1060 '*
1070 '*****
1080 '
1090 'SETUP MEMORY
1100 PCLEAR 1
1110 CLEAR 200, &H1C00
1120 '
1130 'LOAD ML ROUTINES
1140 LOADM "REGXFR.BIN"
1150 LOADM "VIDCLS.BIN"
1160 LOADM "PUTCHR.BIN"
1170 LOADM "GETCHR.BIN"
1180 LOADM "PUTBYT.BIN"
1190 LOADM "SCROLL.BIN"
1200 LOADM "PUTCHA.BIN"
1210 LOADM "TEST0006.BIN"
1220 '
1230 'GO DO THE TEST
1240 EXEC &H7000
1250 '
32767 END
```

Note that I've discontinued the listing of all the Register Transfer Variables in these BASIC Language Control Programs. They were present for information purposes only, and are now taking up too much room and obscuring the more pertinent portions of the programs.

____

Result:



As expected.

=====

PUTBYA: Put an 8-bit Number To the VIDRAM Screen As Two Hexadecimal Digits At the Cursor Position and Advance the Cursor

This routine is designed to function with continuous scrolling output.

PUTBYA advances the cursor, and it also scrolls the screen when required. If those provisions are not necessary for a particular application, use **PUTBYT** instead: it uses less resources.

```
      *****
      *
      * PUTBYA.ASM
      * MDJ 2021/08/30
      *
      * PUTS AN 8-BIT NUMBER
      * TO VIDRAM AS TWO
      * HEXADECIMAL DIGITS
      *
      * SCROLLS THE SCREEN
      * IF REQUIRED
      *
      * ENTRY CONDITIONS:
      *   A = THE 8-BIT NUMBER
      *
      * EXIT CONDITIONS:
      *   NONE
      *
      *****

      * SCRATCHPAD VARIABLES
      * THE 8-BIT NUMBER
0076  L0076   EQU     $0076

      * THE HIGH NIBBLE
0077  L0077   EQU     $0077

      * THE LOW NIBBLE
00F3  L00F3   EQU     $00F3

```

```
                    00400 * EXTERNAL ROUTINE
                    00410 * ADDRESS
          1D36      00420 PUTCHA  EQU     $1D36
                    00430
1D57                00440         ORG     $1D57
                    00450
                    00460 * SAVE THE NUMBER
1D57 97   76        00470 PUTBYA  STA     L0076
                    00480
                    00490 * DIVIDE BY 16
1D59 44             00500         LSRA
1D5A 44             00510         LSRA
1D5B 44             00520         LSRA
1D5C 44             00530         LSRA
                    00540
                    00550 * SAVE THE HIGH NIBBLE
1D5D 97   77        00560         STA     L0077
                    00570
                    00580 * MULTIPLY BY 16
1D5F 48             00590         LSLA
1D60 48             00600         LSLA
1D61 48             00610         LSLA
1D62 48             00620         LSLA
                    00630
                    00640 * SAVE TEMP RESULT
1D63 97   F3        00650         STA     L00F3
                    00660
                    00670 * GET THE NUMBER AGAIN
1D65 96   76        00680         LDA     L0076
                    00690
                    00700 * SUBTRACT TEMP RESULT
1D67 90   F3        00710         SUBA    L00F3
                    00720
                    00730 * SAVE LOW NIBBLE
1D69 97   F3        00740         STA     L00F3
                    00750
                    00760 * IS LOW NIBBLE <= 9
1D6B 81   09        00770         CMPA    #9
                    00780
                    00790 * GO IF NO
1D6D 22   04        00800         BHI     L1D73
                    00810
                    00820 * ADD ZERO OFFSET
1D6F 8B   70        00830         ADDA    #112
1D71 20   02        00840         BRA     L1D75
                    00850
                    00860 * ADD "A" OFFSET
```

```
1D73 8B   37       00870 L1D73   ADDA    #55
                   00880
                   00890 * SAVE LOW NIBBLE CHAR
1D75 97   F3       00900 L1D75   STA     L00F3
                   00910
                   00920 * GET HIGH NIBBLE
1D77 96   77       00930         LDA     L0077
                   00940
                   00950 * IS HIGH NIBBLE <= 9
1D79 81   09       00960         CMPA    #9
                   00970
                   00980 * GO IF NO
1D7B 22   04       00990         BHI     L1D81
                   01000
                   01010 * ADD ZERO OFFSET
1D7D 8B   70       01020         ADDA    #112
1D7F 20   02       01030         BRA     L1D83
                   01040
                   01050 * ADD "A" OFFSET
1D81 8B   37       01060 L1D81   ADDA    #55
                   01070
                   01080 * PUT HIGH NIBBLE CHAR
                   01090 * TO VIDRAM
1D83 17   FFB0     01100 L1D83   LBSR    PUTCHA
                   01110
                   01120 * GET LOW NIBBLE CHAR
1D86 96   F3       01130         LDA     L00F3
                   01140
                   01150 * PUT LOW NIBBLE CHAR
                   01160 * TO VIDRAM
1D88 17   FFAB     01170         LBSR    PUTCHA
                   01180
                   01190 * EXIT
1D8B 39            01200         RTS
          0000     32767         END

00000 TOTAL ERRORS
```

____-

The following Assembly Language Test Routine will be used:

```
00100 *****
00110 *
00120 * TEST0007.ASM
00130 * MDJ 2021/08/30
00140 *
```

```
                    00150 * PUTBYA TEST
                    00160 *
                    00170 * PUTS 128 BYTES
                    00180 * ($00 - $7F) TO THE
                    00190 * SCREEN, BEGINNING AT
                    00200 * THE CURRENT CURSOR
                    00210 * POSITION
                    00220 *
                    00230 *****
                    00240
                    00290 * EXTERNAL ROUTINE
                    00300 * ADDRESS
          1D57      00320 PUTBYA  EQU     $1D57
                    00330
7000                00340         ORG     $7000
                    00350
7000 34   06        00360         PSHS    A,B
                    00370
                    00410 * LOAD THE FIRST
                    00420 * BYTE VALUE
7002 86   00        00430         LDA     #$00
                    00440
                    00490 * SAVE THE BYTE VALUE
7004 1F   89        00500         TFR     A,B
                    00510
                    00520 * GO PUT BYTE TO SCREEN
7006 17   AD4E      00530 L7006   LBSR    PUTBYA
                    00540
                    00550 * ARE WE DONE?
7009 C1   7F        00560         CMPB    #$7F
                    00570
                    00580 * GO IF YES
700B 24   05        00590         BHS     L7012
                    00600
                    00610 * GET NEXT BYTE VALUE
700D 5C             00620         INCB
700E 1F   98        00630         TFR     B,A
7010 20   F4        00640         BRA     L7006
                    00650
                    00700 * EXIT
7012 35   06        00710 L7012   PULS    A,B
7014 39             00720         RTS
          0000      32767         END

00000 TOTAL ERRORS
```

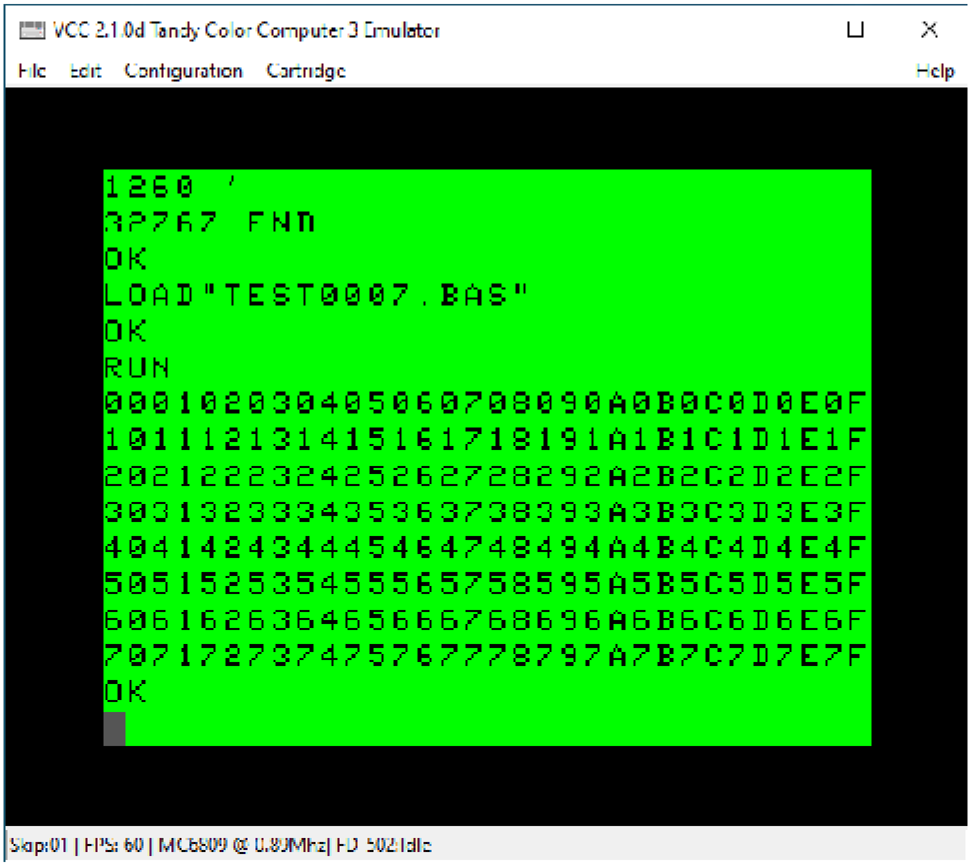
Note Line 560. We make the comparison against Register B, instead of against Register A, because **PUTBYA** alters Register A; i.e. it is not preserved.

We also use the following BASIC Language Control Program:

```
1000 '*****
1010 '*
1020 '* TEST0007.BAS
1030 '* MDJ 2021/08/30
1040 '*
1050 '* PUTBYA TEST
1060 '*
1070 '*****
1080 '
1090 'SETUP MEMORY
1100 PCLEAR 1
1110 CLEAR 200, &H1C00
1120 '
1130 'LOAD ML ROUTINES
1140 LOADM "REGXFR.BIN"
1150 LOADM "VIDCLS.BIN"
1160 LOADM "PUTCHR.BIN"
1170 LOADM "GETCHR.BIN"
1180 LOADM "PUTBYT.BIN"
1190 LOADM "SCROLL.BIN"
1200 LOADM "PUTCHA.BIN"
1210 LOADM "PUTBYA.BIN"
1220 LOADM "TEST0007.BIN"
1230 '
1240 'GO DO THE TEST
1250 EXEC &H7000
1260 '
32767 END
```

First, we perform the test with a simple **RUN** command.

Result:



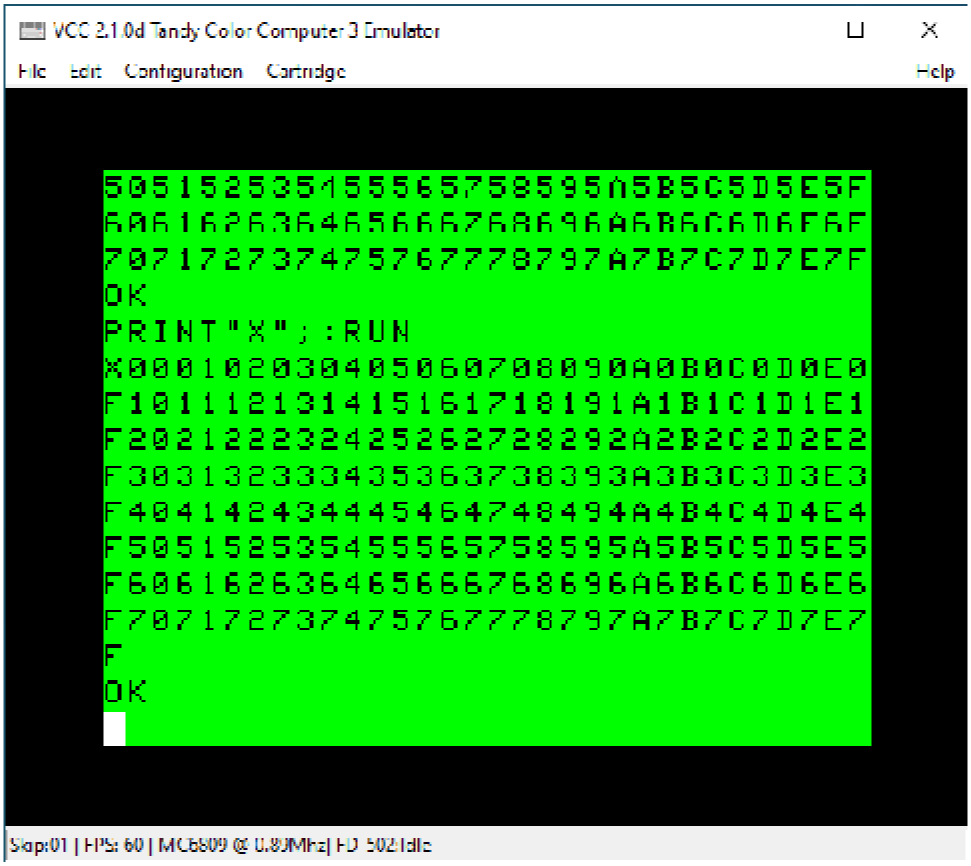
As expected.

But, note that when the scrolls occur, they all occur at the end of the byte.

To test **PUTBYA**'s ability to scroll in the middle of the byte if necessary, we use the command:

```
PRINT"X";:RUN
```

Result:

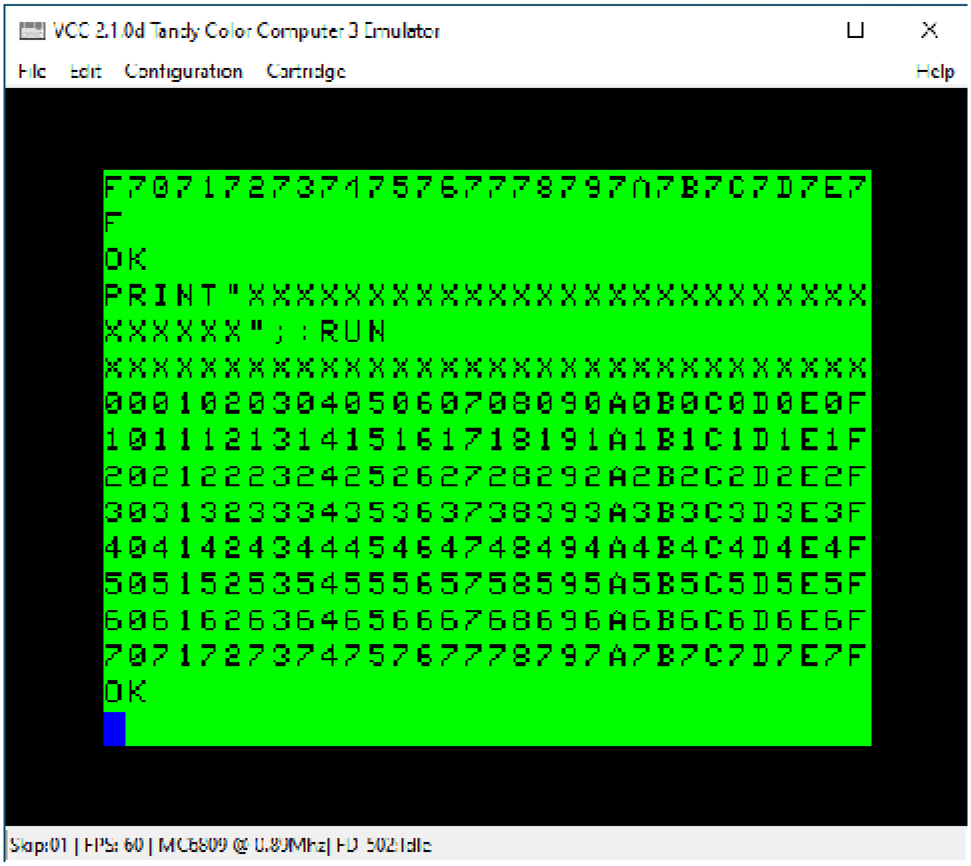


As expected.

And, finally, to test the situation where **PUTBYA** needs to do a scroll before starting to print the byte, we use the command:

PRINT"XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX";:RUN

Result:



As expected.

=====

CRLF: Do a Carriage Return and Line Feed on the VIDRAM Screen

In addition to putting characters and bytes onto the screen, the ability to do a carriage return and line feed will also be useful.

```
                   00100 *****
                   00110 *
                   00120 * CRLF.ASM
                   00130 * MDJ 2021/08/30
                   00140 *
                   00150 * DO A CARRIAGE RETURN
                   00160 * AND LINE FEED
                   00170 *
                   00180 * ENTRY CONDITIONS
                   00190 *   NONE
                   00200 *
                   00210 * EXIT CONDITIONS
                   00220 *   NONE
                   00230 *
                   00240 *****
                   00250
                   00260 * LOW RAM CURSOR ADDRESS
          0088     00270 CURPOS  EQU     $0088
                   00280
                   00290 * SCREEN ADDRESSES
                   00300 * START OF VIDRAM
          0400     00310 VIDRAM  EQU     $0400
                   00320
                   00330 * ONE BYTE PAST THE
                   00340 * END OF VIDRAM
          0600     00350 VIDEND  EQU     $0600
                   00360
                   00370 * START OF THE LAST
                   00380 * LINE OF VIDRAM
          05E0     00390 VIDL15  EQU     $05E0
                   00400
                   00410 * EXTERNAL ROUTINE
                   00420 * ADDRESSES
          1C0E     00430 VIDCLS  EQU     $1C0E
          1D17     00440 SCROLL  EQU     $1D17
                   00450
                   00460
1D8C               00470         ORG     $1D8C
                   00480
```

```
1D8C 34   16         CRLF    PSHS    A,B,X
                     * GET THE CURSOR
1D8E 9E   88                 LDX     CURPOS
                     * IS IT BELOW RANGE?
1D90 8C   0400               CMPX    #VIDRAM
                     * GO IF YES (ERROR)
1D93 25   1D                 BLO     L0002
                     * IS IT ABOVE RANGE?
1D95 8C   0600               CMPX    #VIDEND
                     * GO IF YES (ERROR)
1D98 24   18                 BHS     L0002
                     * IS IT ON THE LAST LINE
                     * OF THE VIDRAM SCREEN?
1D9A 8C   05E0               CMPX    #VIDL15
                     * GO IF YES
1D9D 24   0B                 BHS     L0001
                     * DO THE CRLF
1D9F 1F   10                 TFR     X,D
1DA1 C4   E0                 ANDB    #$E0
1DA3 C3   0020               ADDD    #$0020
1DA6 1F   01                 TFR     D,X
1DA8 20   0E                 BRA     L0003
                     * LAST LINE
1DAA 17   FF6A       L0001   LBSR    SCROLL
1DAD 8E   05E0               LDX     #VIDL15
1DB0 20   06                 BRA     L0003
                     * ERROR
1DB2 17   FE59       L0002   LBSR    VIDCLS
1DB5 8E   0400               LDX     #VIDRAM
                     * PUT THE CURSOR
1DB8 9F   88         L0003   STX     CURPOS
                     * EXIT
1DBA 35   16                 PULS    A,B,X
1DBC 39                      RTS
          0000               END
```

00000 TOTAL ERRORS

Note that if the cursor position has somehow been corrupted and is outside of its proper \$0400 - \$05FF range, **CRLF** simply clears the screen and puts the cursor at its start location, i.e. \$0400.

The actual **CRLF** part of this code, i.e. lines 740-770, might benefit from a little additional explanation. The code transfers the cursor position value from Register **X** to Register **D**, massages it a bit, and then transfers it back to Register **X**.

Each of the sixteen lines on the **VIRRAM** screen begins at a multiple of \$20 (= 032 decimal), i.e. the first character of each line is at the address indicated as follows:

	First Character Address		
Line	**Hexadecimal**	**Decimal**	**Binary**
00	$0400	1024	0000 0100 0000 0000
01	$0420	1056	0000 0100 0010 0000
02	$0440	1088	0000 0100 0100 0000
03	$0460	1120	0000 0100 0110 0000
04	$0480	1152	0000 0100 1000 0000
05	$04A0	1184	0000 0100 1010 0000
06	$04C0	1216	0000 0100 1100 0000
07	$04E0	1248	0000 0100 1110 0000
08	$0500	1280	0000 0101 0000 0000
09	$0520	1312	0000 0101 0010 0000
10	$0540	1344	0000 0101 0100 0000
11	$0560	1376	0000 0101 0110 0000
12	$0580	1408	0000 0101 1000 0000
13	$05A0	1440	0000 0101 1010 0000
14	$05C0	1472	0000 0101 1100 0000
15	$05E0	1504	0000 0101 1110 0000

And, for reference:

VIDRAM	$0400	1024	0000 0100 0000 0000
VIDL01	$0420	1056	0000 0100 0010 0000
VIDL15	$05E0	1472	0000 0101 1110 0000
VIDEND	$0600	1536	0110 0000 0000 0000

Notice that the last five bits in the binary format are all zeroes. **2^5 = 32 = $20.**

$10000 - $20 = $FFE0 = 1111 1111 1110 0000 binary.

Therefore, if we **AND** any screen address (**$0400 - $05FF**) with **$FFE0**, we will obtain the address of the first character of the line in which that screen address appears.

If we then add **32 = $20** to that value, we will obtain the address of the first character of the following line.

So, our logical approach would then simply seem to be:

```
ANDD    $FFE0
```

Unfortunately, there is no **ANDD** in MC6809 machine language; only **ANDA**, **ANDB**, and **ANDCC**. However, since Register A is the high byte of Register D and Register B is the low byte of Register D, i.e.:

```
D = A:B
```

We can accomplish the same task by doing:

```
ANDA    $FF
ANDB    $E0
```

and, because **ANDA $FF** will always leave Register A unchanged, we can simply leave that line of code out. Thus **ANDB $E0** will give us the address of the first character of the current line in Register D, and adding $0020 to that will give us the address of the first character in the following line. And so we have:

```
00730 * DO THE CRLF
00740          TFR     X,D
00750          ANDB    #$E0
00760          ADDD    #$0020
00770          TFR     D,X
```

——-

The Assembly Language Test Routine:

```
00100 *****
00110 *
00120 * TEST0008.ASM
00130 * MDJ 2021/08/31
00140 *
00150 * CRLF TEST
```

```
                  00160 *
                  00170 *****
                  00180
                  00190 * EXTERNAL ROUTINE
                  00200 * ADDRESSES
          1D36    00210 PUTCHA  EQU     $1D36
          1D8C    00220 CRLF    EQU     $1D8C
                  00230
7000              00240         ORG     $7000
                  00250
7000 34   20      00260         PSHS    Y
7002 20   0F      00270         BRA     L7013
                  00280
7004      47      00290 STR01   FCC     /GOD/
          4F
          44
7007      00      00300         FCB     $00
7008      4C      00310 STR02   FCC     /LOVES/
          4F
          56
          45
          53
700D      60      00320         FCB     $60
700E      59      00330         FCC     /YOU/
          4F
          55
7011      61      00340         FCB     $61
7012      00      00350         FCB     $00
                  00360
                  00370 * POINT TO FIRST STRING
7013 108E 7004    00380 L7013   LDY     #STR01
                  00390
                  00400 * GET A CHARACTER
7017 A6   A0      00410 L7017   LDA     ,Y+
                  00420
                  00430 * GO IF NULL TERMINATOR
7019 27   05      00440         BEQ     L7020
                  00450
                  00460 * PUT IT TO VIDRAM
701B 17   AD18    00470         LBSR    PUTCHA
701E 20   F7      00480         BRA     L7017
                  00490
                  00500 * GO DO THE CRLF
7020 17   AD69    00510 L7020   LBSR    CRLF
                  00520
                  00530 * POINT TO SECOND STRING
7023 108E 7008    00540         LDY     #STR02
```

```
                  00550
                  00560 * GET A CHARACTER
7027 A6   A0      00570 L7027   LDA     ,Y+
                  00580
                  00590 * GO IF NULL TERMINATOR
7029 27   05      00600         BEQ     L7030
                  00610
                  00620 * PUT IT TO VIDRAM
702B 17   AD08    00630         LBSR    PUTCHA
702E 20   F7      00640         BRA     L7027
                  00650
                  00660 * EXIT
7030 35   20      00670 L7030   PULS    Y
7032 39           00680         RTS
          0000    32767         END

00000 TOTAL ERRORS
```

And the BASIC Language Control Program:

```
1000 '*****
1010 '*
1020 '* TEST0008.BAS
1030 '* MDJ 2021/08/31
1040 '*
1050 '* CRLF TEST
1060 '*
1070 '*****
1080 '
1090 'SETUP MEMORY
1100 PCLEAR 1
1110 CLEAR 200, &H1C00
1120 '
1130 'LOAD ML ROUTINES
1140 LOADM "REGXFR.BIN"
1150 LOADM "VIDCLS.BIN"
1160 LOADM "PUTCHR.BIN"
1170 LOADM "GETCHR.BIN"
1180 LOADM "PUTBYT.BIN"
1190 LOADM "SCROLL.BIN"
1200 LOADM "PUTCHA.BIN"
1210 LOADM "PUTBYA.BIN"
1220 LOADM "CRLF.BIN"
1230 LOADM "TEST0008.BIN"
1240 '
1250 'GO DO THE TEST
1260 EXEC &H7000
```

```
1270 '
32767 END
```

——-

Result:

As expected.

=====

PK2PRT: Converting POKE Codes to PRINT Codes

The Video Screen VIDRAM displays text characters and Semigraphics characters depending upon which bytes are put to memory in the VIDRAM space ($0400-$05FF).

In the BASIC Language, as discussed in (MDJ01), for each byte ($00-$FF), the character displayed depends upon the method (PRINT or POKE) by which the byte is put to the VIDRAM.

In Assembly Language (and thus also in The ML Foundation), the method used (STA) is analogous to BASIC's POKE mechanism. But the Key Codes returned from the keyboard are PRINT codes. Thus, we need a means for converting between the two types of Key Codes.

In performing such conversions, you may expect that PRT2PK (See next Section) will be required significantly more often than PK2PRT which is discussed in this Section.

If you have a POKE Code, and you want to use it in a PRINT function, convert it to a PRINT Code. If:

> 000 <= POKE Code <= 031 ($00 <= POKE Code <= $1F)
> Then add 096 ($60) to the POKE Code.
>
> 032 <= POKE Code <= 063 ($20 <= POKE Code <= $3F)
> Then, there is no PRINT Code that Corresponds to that POKE Code;
> Use PRINT Code = 32 ($20) = a blank green space.
>
> 064 <= POKE Code <= 095 ($40 <= POKE Code <= $5F)
> Then the PRINT Code is the same as the POKE Code.
>
> 096 <= POKE Code <= 127 ($60 <= POKE Code <= $7F)
> Then subtract 064 ($40) from the POKE Code.
>
> 128 <= POKE Code <= 255) ($80 <= POKE Code <= $FF)
> Then the PRINT Code is the same as the POKE Code.

```
*****
*
* PK2PRT.ASM
* MDJ 2021/08/28
*
* CONVERTS POKE CODES
* TO PRINT CODES
*
* ENTRY CONDITIONS:
```

```
                   00190 *   A = POKE CODE
                   00200 *       ($00 - $FF)
                   00210 *       (000 - 255)
                   00220 *
                   00230 * EXIT CONDITIONS:
                   00240 *   A = PRINT CODE
                   00250 *       ($00 - $FF)
                   00260 *       (000 - 255)
                   00270 *
                   00280 *****
                   00290
1DBD               00300         ORG     $1DBD
                   00310
                   00320 * 000 <= CODE <= 031 ?
1DBD 81   20       00330 PK2PRT  CMPA    #32
                   00340
                   00350 * GO IF YES
1DBF 25   16       00360         BLO     L1DD7
                   00370
                   00380 * 032 <= CODE <= 063 ?
1DC1 81   40       00390         CMPA    #64
                   00400
                   00410 * GO IF YES
1DC3 25   0E       00420         BLO     L1DD3
                   00430
                   00440 * 064 <= CODE <= 095 ?
1DC5 81   60       00450         CMPA    #96
                   00460
                   00470 * GO IF YES ==> NO CHANGE
1DC7 25   10       00480         BLO     L1DD9
                   00490
                   00500 * 096 <= CODE <= 127 ?
1DC9 81   80       00510         CMPA    #128
                   00520
                   00530 * GO IF YES
1DCB 25   02       00540         BLO     L1DCF
                   00550
                   00560 * CODE >= 128
                   00561 * NO CHANGE
1DCD 20   0A       00570         BRA     L1DD9
                   00580
                   00590 * 096 <= CODE <= 127
1DCF 80   40       00600 L1DCF   SUBA    #64
1DD1 20   06       00610         BRA     L1DD9
                   00620
                   00630 * 064 <= CODE <= 095
                   00640 * NO CHANGE
```

```
                   00650
                   00660 * 032 <= CODE <= 063
                   00670 * ALL = A GREEN SPACE
1DD3 86   20       00680 L1DD3   LDA     #32
1DD5 20   02       00690         BRA     L1DD9
                   00700
                   00710 * 000 <= CODE <= 031
1DD7 8B   60       00720 L1DD7   ADDA    #96
                   00730
                   00740 * EXIT
1DD9 39            00750 L1DD9   RTS
          0000     32767         END

00000 TOTAL ERRORS
```

——-

The Assembly Language Test Routine:

```
                   00100 *****
                   00110 *
                   00120 * TEST0009.ASM
                   00130 * MDJ 2021/08/31
                   00140 *
                   00150 * PK2PRT TEST
                   00160 *
                   00170 *****
                   00180
                   00190 * EXTERNAL ROUTINE
                   00200 * ADDRESSES
          1D36     00210 PUTCHA  EQU     $1D36
          1D57     00220 PUTBYA  EQU     $1D57
          1D8C     00230 CRLF    EQU     $1D8C
          1DBD     00240 PK2PRT  EQU     $1DBD
                   00250
7000               00260         ORG     $7000
                   00270
7000 34   20       00280         PSHS    Y
7002 20   17       00290         BRA     L701B
                   00300
                   00310 * OUTPUT STRINGS LIST
7004      60       00320 STR01   FCB     $60
7005      50       00330         FCC     /POKE/
          4F
          4B
          45
7009      60       00340         FCB     $60
```

```
700A      00     00350          FCB    $00
700B      50     00360 STR02    FCC    /PRINT/
          52
          49
          4E
          54
7010      60     00370          FCB    $60
7011      00     00380          FCB    $00
7012      43     00390 STR03    FCC    /CODE/
          4F
          44
          45
7016      60     00400          FCB    $60
7017      7D     00410          FCB    $7D
7018      60     00420          FCB    $60
7019      64     00430          FCB    $64
701A      00     00440          FCB    $00
                 00450
                 00460 * MAIN ROUTINE
701B 86   12     00470 L701B    LDA    #18
701D 8D   0E     00480          BSR    L702D
701F 86   28     00490          LDA    #40
7021 8D   0A     00500          BSR    L702D
7023 86   57     00510          LDA    #87
7025 8D   06     00520          BSR    L702D
7027 86   77     00530          LDA    #119
7029 8D   02     00540          BSR    L702D
702B 20   3D     00550          BRA    L706A
                 00560
                 00570 * PRINT ORDER SUBROUTINE
                 00580 * SAVE POKE CODE
702D 34   02     00590 L702D    PSHS   A
                 00600
                 00610 * PUT POKE MESSAGE
702F 8D   1F     00620          BSR    L7050
7031 8D   29     00630          BSR    L705C
                 00640
                 00650 * RESTORE AND RE-SAVE
                 00660 * POKE CODE
7033 35   02     00670          PULS   A
7035 34   02     00680          PSHS   A
                 00690
                 00700 * PUT POKE CODE
7037 17   AD1D   00710          LBSR   PUTBYA
                 00720
                 00730 * PUT PRINT MESSAGE
703A 17   AD4F   00740          LBSR   CRLF
```

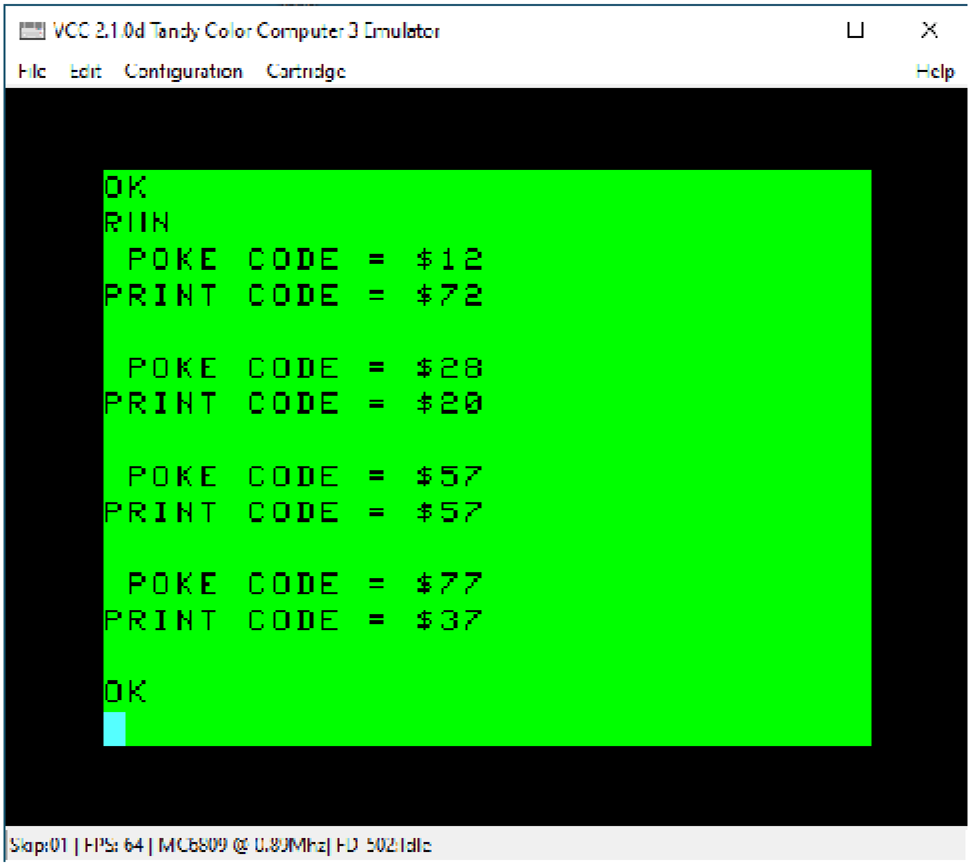
```
703D 8D   17       00750         BSR    L7056
703F 8D   1B       00760         BSR    L705C
                   00770
                   00780 * RESTORE POKE CODE
7041 35   02       00790         PULS   A
                   00800
                   00810 * CONVERT CODE
7043 17   AD77     00820         LBSR   PK2PRT
                   00830
                   00840 * PUT PRINT CODE
7046 17   AD0E     00850         LBSR   PUTBYA
7049 17   AD40     00860         LBSR   CRLF
704C 17   AD3D     00870         LBSR   CRLF
704F 39            00880         RTS
                   00890
                   00900 * PRINT MESSAGE SUBRT
7050 108E 7004     00910 L7050   LDY    #STR01
7054 20   0A       00920         BRA    L7060
7056 108E 700B     00930 L7056   LDY    #STR02
705A 20   04       00940         BRA    L7060
705C 108E 7012     00950 L705C   LDY    #STR03
7060 A6   A0       00960 L7060   LDA    ,Y+
7062 27   05       00970         BEQ    L7069
7064 17   ACCF     00980         LBSR   PUTCHA
7067 20   F7       00990         BRA    L7060
7069 39            01000 L7069   RTS
                   01010
                   01020 * EXIT
706A 35   20       01030 L706A   PULS   Y
706C 39            01040         RTS
          0000     32767         END

00000 TOTAL ERRORS
```

The BASIC Language Control Program:

```
1000 '*****
1010 '*
1020 '* TEST0009.BAS
1030 '* MDJ 2021/09/01
1040 '*
1050 '* PK2PRT TEST
1060 '*
1070 '*****
1080 '
1090 'SETUP MEMORY
```

```
1100 PCLEAR 1
1110 CLEAR 200, &H1C00
1120 '
1130 'LOAD ML ROUTINES
1140 LOADM "REGXFR.BIN"
1150 LOADM "VIDCLS.BIN"
1160 LOADM "PUTCHR.BIN"
1170 LOADM "GETCHR.BIN"
1180 LOADM "PUTBYT.BIN"
1190 LOADM "SCROLL.BIN"
1200 LOADM "PUTCHA.BIN"
1210 LOADM "PUTBYA.BIN"
1220 LOADM "CRLF.BIN"
1230 LOADM "PK2PRT.BIN"
1240 LOADM "PRT2PK.BIN"
1250 LOADM "TEST0009.BIN"
1260 '
1270 'GO DO THE TEST
1280 EXEC &H7000
1290 '
32767 END
```

Result:



As expected.

=====

PRT2PK: Converting PRINT Codes to POKE Codes

The Video Screen VIDRAM displays text characters and Semigraphics characters depending upon which bytes are put to memory in the VIDRAM space ($0400-$05FF).

In the BASIC Language, as discussed in (MDJ01), for each byte ($00-$FF), the character displayed depends upon the method (PRINT or POKE) by which the byte is put to the VIDRAM.

In Assembly Language (and thus also in The ML Foundation), the method used (STA) is analogous to BASIC's POKE mechanism. But the Key Codes returned from the keyboard are PRINT codes. Thus, we need a means for converting between the two types of Key Codes.

In performing such conversions, you may expect that PRT2PK will be required significantly more often than PK2PRT (See previous Section) .

If you have a PRINT Code, and you want to use it in a POKE function, convert it to a POKE Code. If:

000 <= PRINT Code <= 031 ($00 <= PRINT Code <= $1F)
Then it is a blank background color space.
Use POKE Code = 096 ($60) = a blank green space.

032 <= PRINT Code <= 063 ($20 <= PRINT Code <= $3F)
Then add 064 ($40) to the PRINT Code.

064 <= PRINT Code <= 095 ($40 <= PRINT Code <= $5F)
Then the POKE Code is the same as the PRINT Code.

096 <= PRINT Code <= 127 ($60 <= PRINT Code <= $7F)
Then subtract 096 ($60) from the PRINT Code.

128 <= PRINT Code <= 255) ($80 <= PRINT Code <= $FF)
Then the POKE Code is the same as the PRINT Code.

```
00100 *****
00110 *
00120 * PRT2PK.ASM
00130 * MDJ 2021/08/28
00140
00150 * CONVERTS PRINT CODES
00160 * TO POKE CODES
00170 *
00180 * ENTRY CONDITIONS
```

```
                   00190 *   A = PRINT CODE
                   00200 *       ($00 - $FF)
                   00210 *       (000 - 255)
                   00220 *
                   00230 * EXIT CONDITIONS
                   00240 *   A = POKE CODE
                   00250 *       ($00 - $FF)
                   00260 *       (000 - 255)
                   00270 *
                   00280 *****
                   00290
1DDA               00300         ORG     $1DDA
                   00310
                   00320 * 000 <= CODE <= 031 ?
1DDA 81   20       00330 PRT2PK  CMPA    #32
                   00340
                   00350 * GO IF YES
1DDC 25   16       00360         BLO     L1DF4
                   00370
                   00380 * 032 <= CODE <= 063 ?
1DDE 81   40       00390         CMPA    #64
                   00400
                   00410 * GO IF YES
1DE0 25   0E       00420         BLO     L1DF0
                   00430
                   00440 * 064 <= CODE <= 095 ?
1DE2 81   60       00450         CMPA    #96
                   00460
                   00470 * GO IF YES ==> NO CHANGE
1DE4 25   10       00480         BLO     L1DF6
                   00490
                   00500 * 096 <= CODE <= 127 ?
1DE6 81   80       00510         CMPA    #128
                   00520
                   00530 * GO IF YES
1DE8 25   02       00540         BLO     L1DEC
                   00550
                   00560 * CODE >= 128
                   00570 * NO CHANGE
1DEA 20   0A       00580         BRA     L1DF6
                   00590
                   00600 * 096 <= CODE <= 127
1DEC 80   60       00610 L1DEC   SUBA    #96
1DEE 20   06       00620         BRA     L1DF6
                   00630
                   00640 * 064 <= CODE <= 095
                   00650 * NO CHANGE
```

```
                   00660
                   00670 * 032 <= CODE <= 063
1DF0 8B   40       00680 L1DF0   ADDA    #64
1DF2 20   02       00690         BRA     L1DF6
                   00700
                   00710 * 000 <= CODE <= 031
                   00720 * ALL = A GREEN SPACE
1DF4 86   60       00730 L1DF4   LDA     #96
                   00740
                   00750 * EXIT
1DF6 39            00760 L1DF6   RTS
          0000     32767         END

00000 TOTAL ERRORS
```

——-

The Assembly Language Test Routine:

```
                   00100 *****
                   00110 *
                   00120 * TEST0010.ASM
                   00130 * MDJ 2021/08/31
                   00140 *
                   00150 * PRT2PK TEST
                   00160 *
                   00170 *****
                   00180
                   00190 * EXTERNAL ROUTINE
                   00200 * ADDRESSES
          1D36     00210 PUTCHA  EQU     $1D36
          1D57     00220 PUTBYA  EQU     $1D57
          1D8C     00230 CRLF    EQU     $1D8C
          1DDA     00240 PRT2PK  EQU     $1DDA
                   00250
7000               00260         ORG     $7000
                   00270
7000 34   20       00280         PSHS    Y
7002 20   17       00290         BRA     L701B
                   00300
                   00310 * OUTPUT STRINGS LIST
7004      50       00320 STR01   FCC     /PRINT/
          52
          49
          4E
          54
7009      60       00330         FCB     $60
```

```
700A      00     00340          FCB     $00
700B      60     00350 STR02    FCB     $60
700C      50     00360          FCC     /POKE/
          4F
          4B
          45
7010      60     00370          FCB     $60
7011      00     00380          FCB     $00
7012      43     00390 STR03    FCC     /CODE/
          4F
          44
          45
7016      60     00400          FCB     $60
7017      7D     00410          FCB     $7D
7018      60     00420          FCB     $60
7019      64     00430          FCB     $64
701A      00     00440          FCB     $00
                 00450
                 00460 * MAIN ROUTINE
701B 86   12     00470 L701B    LDA     #18
701D 8D   0E     00480          BSR     L702D
701F 86   28     00490          LDA     #40
7021 8D   0A     00500          BSR     L702D
7023 86   57     00510          LDA     #87
7025 8D   06     00520          BSR     L702D
7027 86   77     00530          LDA     #119
7029 8D   02     00540          BSR     L702D
702B 20   3D     00550          BRA     L706A
                 00560
                 00570 * PRINT ORDER SUBROUTINE
                 00580 * SAVE PRINT CODE
702D 34   02     00590 L702D    PSHS    A
                 00600
                 00610 * PUT PRINT MESSAGE
702F 8D   1F     00620          BSR     L7050
7031 8D   29     00630          BSR     L705C
                 00640
                 00650 * RESTORE AND RE-SAVE
                 00660 * PRINT CODE
7033 35   02     00670          PULS    A
7035 34   02     00680          PSHS    A
                 00690
                 00700 * PUT PRINT CODE
7037 17   AD1D   00710          LBSR    PUTBYA
                 00720
                 00730 * PUT POKE MESSAGE
703A 17   AD4F   00740          LBSR    CRLF
```

```
703D 8D   17       00750          BSR     L7056
703F 8D   1B       00760          BSR     L705C
                   00770
                   00780 * RESTORE PRINT CODE
7041 35   02       00790          PULS    A
                   00800
                   00810 * CONVERT CODE
7043 17   AD94     00820          LBSR    PRT2PK
                   00830
                   00840 * PUT POKE CODE
7046 17   AD0E     00850          LBSR    PUTBYA
7049 17   AD40     00860          LBSR    CRLF
704C 17   AD3D     00870          LBSR    CRLF
704F 39            00880          RTS
                   00890
                   00900 * PRINT MESSAGE SUBRT
7050 108E 7004     00910 L7050    LDY     #STR01
7054 20   0A       00920          BRA     L7060
7056 108E 700B     00930 L7056    LDY     #STR02
705A 20   04       00940          BRA     L7060
705C 108E 7012     00950 L705C    LDY     #STR03
7060 A6   A0       00960 L7060    LDA     ,Y+
7062 27   05       00970          BEQ     L7069
7064 17   ACCF     00980          LBSR    PUTCHA
7067 20   F7       00990          BRA     L7060
7069 39            01000 L7069    RTS
                   01010
                   01020 * EXIT
706A 35   20       01030 L706A    PULS    Y
706C 39            01040          RTS
          0000     32767          END

00000 TOTAL ERRORS
```

The BASIC Language Control Program:

```
1000 '*****
1010 '*
1020 '* TEST0010.BAS
1030 '* MDJ 2021/09/01
1040 '*
1050 '* PRT2PK TEST
1060 '*
1070 '*****
1080 '
1090 'SETUP MEMORY
1100 PCLEAR 1
```

```
1110 CLEAR 200, &H1C00
1120 '
1130 'LOAD ML ROUTINES
1140 LOADM "REGXFR.BIN"
1150 LOADM "VIDCLS.BIN"
1160 LOADM "PUTCHR.BIN"
1170 LOADM "GETCHR.BIN"
1180 LOADM "PUTBYT.BIN"
1190 LOADM "SCROLL.BIN"
1200 LOADM "PUTCHA.BIN"
1210 LOADM "PUTBYA.BIN"
1220 LOADM "CRLF.BIN"
1230 LOADM "PK2PRT.BIN"
1240 LOADM "PRT2PK.BIN"
1250 LOADM "TEST0010.BIN"
1260 '
1270 'GO DO THE TEST
1280 EXEC &H7000
1290 '
32767 END
```

Result:

As expected.

=====

POLCAT: Get a Key Press Character Code From the Keyboard

With the completion of this **`POLCAT`** Routine, we will have a minimally complete system; capable of receiving input from the Keyboard and generating output to the **`VIDRAM`** Screen.

In conformance with my CoCo Philosophy of trying to cram as much stuff as I can into the 96K of the 64K CoCo 2, **`POLCAT`** simply jumps into ROM and uses it's POLCAT Routine. (cf. Appendix A below for the details of my CoCo Philosophy).

I developed all the other Routines presented in this Paper entirely in Assembly Language in order to maximize speed while also minimizing memory space used.

But, with the Keyboard, the limitations of the BASIC Language (Very Slow) are enormously overshadowed by the User Limitations (SUPER HUMONGOUS SLOW). There is thus no point in trying to write super-fast machine code for Keyboard access. It is much more efficient to just use the ROM code: it immensely saves on RAM space, and won't materially effect overall system speed in normal use.

In the future, I will use the same approach regarding access to comparatively slow peripherals such as the Cassette and Disk Drives, and the RS-232 Port.

So, here is the final section of code being presented herein:

```
00100 *****
00110 *
00120 * POLCAT.ASM
00130 * MDJ 1990/09/25
00140 *   AS REFORMATTED
00150 *   2021/08/27
00160 *
00170 * POLLS THE KEYBOARD
00180 * FOR A KEYSTROKE
00190 *
00200 * USES THE BUILT-IN
00210 * ROM POLCAT ROUTINE
00220 *
00230 * ENTRY CONDITIONS:
00240 *   NONE
00250 *
00260 * EXIT CONDITIONS:
00270 *  IF NO KEY WAS PRESSED:
00280 *   CC Z BIT = 1
00290 *   REG A    = 0
```

```
                    00300 *  IF A KEY WAS PRESSED
                    00310 *   CC Z BIT = 0
                    00320 *   REG A    = CHAR CODE
                    00330 *
                    00340 * ANY ROUTINE CALLING
                    00350 * THIS SHOULD PSHS A,CC
                    00360 * BEFORE CALLING AND
                    00370 * PULS A,CC ON RETURN
                    00380 *
                    00390 *****
                    00400
                    00410 * RAMROM TRIGGER ADDRESS
          FFDE      00420 RAMROM  EQU     $FFDE
                    00430
                    00440 * ALLRAM TRIGGER ADDRESS
          FFDF      00450 ALLRAM  EQU     $FFDF
                    00460
                    00470 * ROM POLCAT JUMP ADDRESS
          A000      00480 XPOLCT  EQU     $A000
                    00490
1DF7                00500         ORG     $1DF7
                    00510
1DF7 34   68        00520 POLCAT  PSHS    Y,U,DP
                    00530
                    00540 * SET RAMROM MODE
1DF9 B7   FFDE      00550         STA     RAMROM
                    00560
                    00570 * GO DO ROM POLCAT
1DFC AD   9F A000   00580         JSR     [XPOLCT]
                    00590
                    00600 * SET ALLRAM MODE
1E00 B7   FFDF      00610         STA     ALLRAM
                    00620
                    00630 * EXIT
1E03 35   68        00640         PULS    Y,U,DP
1E05 39             00650         RTS
          0000      32767         END

00000 TOTAL ERRORS
```

——-

The general concept for the Test Routine is simply to receive Key Press Codes from the Keyboard and echo them to the **VIDRAM** Screen.

The CoCo 2 Keyboard is not capable of directly generating Codes 000 - 002, 004 - 007, 011, 014 - 020, 022 - 031, 096, or 123 - 255 ($00 - $02, $04 - $07, $0B, $0E - $14, $16 - $1F, $60, or $7B - $FF).

And, for the purposes of this Test, we will ignore Codes 009, 010, 012, and 021 ($09, $0A, $0C, and $15).

When the Test receives Code 003 ($03), it will do a BREAK (i.e. exit the program) and return to the Command Prompt.

When the Test receives Code 008 ($08), it will perform the Backspace and Overwrite Function.

When the Test receives Code 013 ($0D), it will perform the Carriage Return/Linefeed Function.

For all other Codes, the Test Routine will echo the appropriate character to the **VIDRAM** Screen. The Key Press Character Codes returned from the Keyboard are PRINT Codes rather than POKE Codes. Therefore, any such codes which are to be output to the **VIDRAM** Screen must be processed through **PRT2PK** first.

The Assembly Language Test Routine:

```
                  00100 *****
                  00110 *
                  00120 * TEST0011.ASM
                  00130 * MDJ 2021/09/01
                  00140 *
                  00150 * POLCAT TEST
                  00160 *
                  00170 *****
                  00180
                  00190 * LOW RAM CURSOR ADDRESS
          0088    00200 CURPOS  EQU     $0088
                  00210
                  00220 * EXTERNAL ROUTINE
                  00230 * ADDRESSES
          1C1F    00240 PUTCHR  EQU     $1C1F
          1D36    00250 PUTCHA  EQU     $1D36
          1D8C    00260 CRLF    EQU     $1D8C
          1DDA    00270 PRT2PK  EQU     $1DDA
          1DF7    00280 POLCAT  EQU     $1DF7
                  00290
7000              00300         ORG     $7000
                  00310
7000 34   03      00320         PSHS    A,CC
                  00330
```

```
                   00340 * DISPLAY PROMPT
7002 86   7E       00350       LDA    #126
7004 17   AD2F     00360       LBSR   PUTCHA
7007 86   60       00370       LDA    #96
7009 17   AD2A     00380       LBSR   PUTCHA
                   00390
                   00400 * GO CHECK ROM
                   00410 * FOR KEY PRESS
700C 17   ADE8     00420 L700C LBSR   POLCAT
                   00430
                   00440 * GO IF NO KEY PRESS
700F 27   FB       00450       BEQ    L700C
                   00460
                   00470 * WAS IT THE BREAK KEY?
7011 81   03       00480       CMPA   #3
                   00490
                   00500 * GO IF YES
7013 27   32       00510       BEQ    L7047
                   00520
                   00530 * WAS IT A BACKSPACE?
7015 81   08       00540       CMPA   #8
                   00550
                   00560 * GO IF YES
7017 27   1D       00570       BEQ    L7036
                   00580
                   00590 * WAS IT A CARRIAGE
                   00600 * RETURN?
7019 81   0D       00610       CMPA   #13
                   00620
                   00630 * GO IF YES
701B 27   14       00640       BEQ    L7031
                   00650
                   00660 * WAS IT <= 031 ?
701D 81   20       00670       CMPA   #32
                   00680
                   00690 * GO IF YES (IGNORE)
701F 25   EB       00700       BLO    L700C
                   00710
                   00720 * WAS IT = 96 ?
7021 81   60       00730       CMPA   #96
                   00740
                   00750 * GO IF YES (IGNORE)
7023 27   E7       00760       BEQ    L700C
                   00770
                   00780 * WAS IT >= 123 ?
7025 81   7B       00790       CMPA   #123
                   00800
```

```
                   00810 * GO IF YES (IGNORE)
7027 24   E3       00820         BHS     L700C
                   00830
                   00840 * PUT IT TO VIDRAM
7029 17   ADAE     00850         LBSR    PRT2PK
702C 17   AD07     00860         LBSR    PUTCHA
702F 20   DB       00870         BRA     L700C
                   00880
                   00890 * DO CRLF
7031 17   AD58     00900 L7031   LBSR    CRLF
7034 20   D6       00910         BRA     L700C
                   00920
                   00930 * DO BACKSPACE
7036 34   02       00940 L7036   PSHS    A
7038 9E   88       00950         LDX     CURPOS
703A 30   1F       00960         LEAX    -1,X
703C 9F   88       00970         STX     CURPOS
703E 86   60       00980         LDA     #96
7040 17   ABDC     00990         LBSR    PUTCHR
7043 35   02       01000         PULS    A
7045 20   C5       01010         BRA     L700C
                   01020
                   01030 * EXIT
7047 35   03       01040 L7047   PULS    A,CC
7049 39            01050         RTS
          0000     32767         END

00000 TOTAL ERRORS
```

The BASIC Language Control Program:

```
1000 '*****
1010 '*
1020 '* TEST0011.BAS
1030 '* MDJ 2021/09/01
1040 '*
1050 '* POLCAT TEST
1060 '*
1070 '*****
1080 '
1090 'SETUP MEMORY
1100 PCLEAR 1
1110 CLEAR 200, &H1C00
1120 '
1130 'LOAD ML ROUTINES
1140 LOADM "REGXFR.BIN"
1150 LOADM "VIDCLS.BIN"
```

```
1160 LOADM "PUTCHR.BIN"
1170 LOADM "GETCHR.BIN"
1180 LOADM "PUTBYT.BIN"
1190 LOADM "SCROLL.BIN"
1200 LOADM "PUTCHA.BIN"
1210 LOADM "PUTBYA.BIN"
1220 LOADM "CRLF.BIN"
1230 LOADM "PK2PRT.BIN"
1240 LOADM "PRT2PK.BIN"
1250 LOADM "POLCAT.BIN"
1260 LOADM "TEST0011.BIN"
1270 '
1280 'GO DO THE TEST
1290 EXEC &H7000
1300 '
32767 END
```

——

Result:

As expected, complete with minor typing error: Can you find it?

Ain’t computers grand? They STILL do EXACTLY what you tell them to do.

=====

Bonus Code: Skeletons

Here, and in the following few sections, are some little pieces of Bonus Code beyond the ML Foundation Core for you to play with as you choose.

These new programs are set to initially assemble at $1E06 for testing; and then to be reassembled elsewhere as you see fit. They also tend to use generic labels such as **`L0001, L0002, L0003,`** etc, which you may also choose to revise for your own purposes.

The Assembly Language Test Routines are set to assemble at $7000, and the BASIC Control Programs call the Test Routines at that address.

First of all, here are three pieces of generic Skeleton code which can be used to build and test additions to this system. First, the Skeleton for the Assembly Language Routine to be Developed:

```
00100 *****
00110 *
00120 * PROGSKEL.ASM
00130 * MDJ 20XX/XX/XX
00140 *
00150 * DESCRIPTION:
00160
01000 *
01010 *****
01020
01030 * LOW RAM CURSOR ADDRESS
01040 CURPOS  EQU     $0088
01050
01060 * SCREEN ADDRESSES
01070 * START OF VIDRAM
01080 VIDRAM  EQU     $0400
01090
01100 * ONE BYTE PAST THE
01110 * END OF VIDRAM
01120 VIDEND  EQU     $0600
01130
01140 * ML FOUNDATION
01150 * CORE ADDRESSES
01160 REGXFR  EQU     $1C00
01170 VIDCLS  EQU     $1C0E
01180 PUTCHR  EQU     $1C1F
01190 GETCHR  EQU     $1CD2
01200 PUTBYT  EQU     $1CD5
01210 SCROLL  EQU     $1D17
01220 PUTCHA  EQU     $1D36
```

```
01230 PUTBYA  EQU     $1D57
01240 CRLF    EQU     $1D8C
01250 PK2PRT  EQU     $1DBD
01260 PRT2PK  EQU     $1DDA
01270 POLCAT  EQU     $1DF7
01280
01290 * FOR REFERENCE ONLY:
01300 * ONE BYTE PAST THE
01310 * END OF THE
01320 * ML FOUNDATION CORE
01330 COREND  EQU     $1E06
01340
01350 * ADDITIONAL EQUATES HERE
01360
02000 * PROGRAM BEING DEVELOPED
02010 * PROGRAM ORIGIN:
02020 * (CHANGE AS NEEDED)
02030         ORG     $1E06
02040
32000
32010 * EXIT
32020         RTS
32767         END
```

Add any additional Equates you need after Line 1350, and add your new code after Line 2030.

The Skeleton Assembly Language Test Routine:

```
00100 *****
00110 *
00120 * TESTSKEL.ASM
00130 * MDJ 20XX/XX/XX
00140 *
00150 * TEST OF
00160 * PROGSKEL.ASM
00170
01000 *
01010 *****
01020
01030 * LOW RAM CURSOR ADDRESS
01040 CURPOS  EQU     $0088
01050
01060 * SCREEN ADDRESSES
01070 * START OF VIDRAM
```

```
01080 VIDRAM  EQU     $0400
01090
01100 * ONE BYTE PAST THE
01110 * END OF VIDRAM
01120 VIDEND  EQU     $0600
01130
01140 * ML FOUNDATION
01150 * CORE ADDRESSES
01160 REGXFR  EQU     $1C00
01170 VIDCLS  EQU     $1C0E
01180 PUTCHR  EQU     $1C1F
01190 GETCHR  EQU     $1CD2
01200 PUTBYT  EQU     $1CD5
01210 SCROLL  EQU     $1D17
01220 PUTCHA  EQU     $1D36
01230 PUTBYA  EQU     $1D57
01240 CRLF    EQU     $1D8C
01250 PK2PRT  EQU     $1DBD
01260 PRT2PK  EQU     $1DDA
01270 POLCAT  EQU     $1DF7
01280
01290 * FOR REFERENCE ONLY:
01300 * ONE BYTE PAST THE
01310 * END OF THE
01320 * ML FOUNDATION CORE
01330 COREND  EQU     $1E06
01340
01350 * ADDITIONAL EQUATES HERE
01360
01800 * PROGRAM BEING DEVELOPED
01810 * EQUATE - CHANGE AS
01820 * NEEDED
01830 PRGSKL  EQU     $1E06
01840
02000 * TEST ROUTINE
02010 * PROGRAM ORIGIN:
02020 * (CHANGE AS NEEDED)
02030         ORG     $7000
02040
32000
32010 * EXIT
32020         RTS
32767         END
```

Add any additional Equates you need after Line 1350, add the Equate for the Program you're developing by modifying Line 1830, and add your new Test Routine code after Line 2030.

The Skeleton BASIC Language Control Program:

```
1000 '*****
1010 '*
1020 '* TESTSKEL.BAS
1030 '* MDJ 20XX/XX/XX
1040 '*
1050 '* TEST OF
1060 '* PROGSKEL.ASM
1070 '*
1080 '*****
1090 '
1100 'SETUP MEMORY
1110 PCLEAR 1
1120 CLEAR 200, &H1C00
1130 '
1140 'LOAD ML FOUNDATION
1150 'CORE ROUTINES
1160 LOADM "REGXFR.BIN"
1170 LOADM "VIDCLS.BIN"
1180 LOADM "PUTCHR.BIN"
1190 LOADM "GETCHR.BIN"
1200 LOADM "PUTBYT.BIN"
1210 LOADM "SCROLL.BIN"
1220 LOADM "PUTCHA.BIN"
1230 LOADM "PUTBYA.BIN"
1240 LOADM "CRLF.BIN"
1250 LOADM "PK2PRT.BIN"
1260 LOADM "PRT2PK.BIN"
1270 LOADM "POLCAT.BIN"
1280 '
1290 'LOAD OTHER
1300 'NEEDED ROUTINES
1310 '
2000 'LOAD THE ROUTINE
2010 'BEING DEVELOPED
2020 LOADM "PROGSKEL.BIN"
2030 '
2040 'LOAD THE TEST ROUTINE
2050 LOADM :TESTSKEL.BIN"
2060 '
2070 'GO DO THE TEST
2080 EXEC &H7000
2090 '
```

```
32767 END
```

`LOADM` any additional routines you need after Line 1300, **`LOADM`** the new Program you're developing by modifying Line 2020, and **`LOADM`** the new Test Routine you'll be using by modifying Line 2050.

Please note that because of the "**`1120 CLEAR 200, &H1C00`**" statement, these BASIC Control Programs have very little room to do anything beyond loading Assembly Language Routines and then calling the Test Routine. Accordingly, all your testing and reporting would best be performed in the Assembly Language Test Routine (Of course, "Your Mileage May Vary).

=====

PUTWRA: Put a 16-bit Number To the VIDRAM Screen As Four Hexadecimal Digits At the Cursor Position and Advance the Cursor

In the same way that an 8-bit number is called a byte, a 16-bit number is called a word. In the next section after this one, the PUTWRD Routine is presented. The mnemonic PUTWRD simply stands for “Put Word”. In this section, PUTWRA stands for “Put Word with Advance”.

```
             *****
             *
             * PUTWRA.ASM
             * MDJ 2021/09/09
             *
             * PUTS A 16-BIT NUMBER
             * TO VIDRAM AS FOUR
             * HEXADECIMAL DIGITS
             *
             * ADVANCES THE CURSOR;
             * SCROLLS THE SCREEN
             * IF REQUIRED
             *
             * ENTRY CONDITIONS:
             *   D = THE 16-BIT NUMBER
             *
             * EXIT CONDITIONS:
             *   NONE
             *
             *****

             * LOW RAM CURSOR ADDRESS
0088         CURPOS  EQU     $0088

             * SCREEN ADDRESSES
             * START OF VIDRAM
0400         VIDRAM  EQU     $0400

             * ONE BYTE PAST THE
             * END OF VIDRAM
0600         VIDEND  EQU     $0600

```

```
                    01140 * ML FOUNDATION
                    01150 * CORE ADDRESSES
          1C00      01160 REGXFR  EQU     $1C00
          1C0E      01170 VIDCLS  EQU     $1C0E
          1C1F      01180 PUTCHR  EQU     $1C1F
          1CD2      01190 GETCHR  EQU     $1CD2
          1CD5      01200 PUTBYT  EQU     $1CD5
          1D17      01210 SCROLL  EQU     $1D17
          1D36      01220 PUTCHA  EQU     $1D36
          1D57      01230 PUTBYA  EQU     $1D57
          1D8C      01240 CRLF    EQU     $1D8C
          1DBD      01250 PK2PRT  EQU     $1DBD
          1DDA      01260 PRT2PK  EQU     $1DDA
          1DF7      01270 POLCAT  EQU     $1DF7
                    01280
                    01290 * FOR REFERENCE ONLY:
                    01300 * ONE BYTE PAST THE
                    01310 * END OF THE
                    01320 * ML FOUNDATION CORE
          1E06      01330 COREND  EQU     $1E06
                    01340
1E06                02030         ORG     $1E06
                    02040
                    02050 * SAVE THE LOW BYTE
1E06 34   04        02060         PSHS    B
                    02070
                    02080 * PRINT THE HIGH BYTE
1E08 17   FF4C      02090         LBSR    PUTBYA
                    02100
                    02110 * RESTORE THE LOW BYTE
                    02120 * BUT INTO REGISTER A
1E0B 35   02        02130         PULS    A
                    02140
                    02150 * PRINT THE LOW BYTE
1E0D 17   FF47      02160         LBSR    PUTBYA
                    32000
                    32010 * EXIT
1E10 39             32020         RTS
          0000      32767         END

00000 TOTAL ERRORS
```

The Assembly Language Test Routine:

```
            00100 *****
            00110 *
            00120 * TEST0012.ASM
            00130 * MDJ 2021/09/09
            00140 *
            00150 * TEST OF
            00160 * PUTWRA.ASM
            01000 *
            01010 *****
            01020
            01030 * LOW RAM CURSOR ADDRESS
0088        01040 CURPOS  EQU     $0088
            01050
            01060 * SCREEN ADDRESSES
            01070 * START OF VIDRAM
0400        01080 VIDRAM  EQU     $0400
            01090
            01100 * ONE BYTE PAST THE
            01110 * END OF VIDRAM
0600        01120 VIDEND  EQU     $0600
            01130
            01140 * ML FOUNDATION
            01150 * CORE ADDRESSES
1C00        01160 REGXFR  EQU     $1C00
1C0E        01170 VIDCLS  EQU     $1C0E
1C1F        01180 PUTCHR  EQU     $1C1F
1CD2        01190 GETCHR  EQU     $1CD2
1CD5        01200 PUTBYT  EQU     $1CD5
1D17        01210 SCROLL  EQU     $1D17
1D36        01220 PUTCHA  EQU     $1D36
1D57        01230 PUTBYA  EQU     $1D57
1D8C        01240 CRLF    EQU     $1D8C
1DBD        01250 PK2PRT  EQU     $1DBD
1DDA        01260 PRT2PK  EQU     $1DDA
1DF7        01270 POLCAT  EQU     $1DF7
            01280
            01290 * FOR REFERENCE ONLY:
            01300 * ONE BYTE PAST THE
            01310 * END OF THE
            01320 * ML FOUNDATION CORE
1E06        01330 COREND  EQU     $1E06
            01340
            01800 * PROGRAM BEING DEVELOPED
1E06        01830 PUTWRA  EQU     $1E06
            01840
```

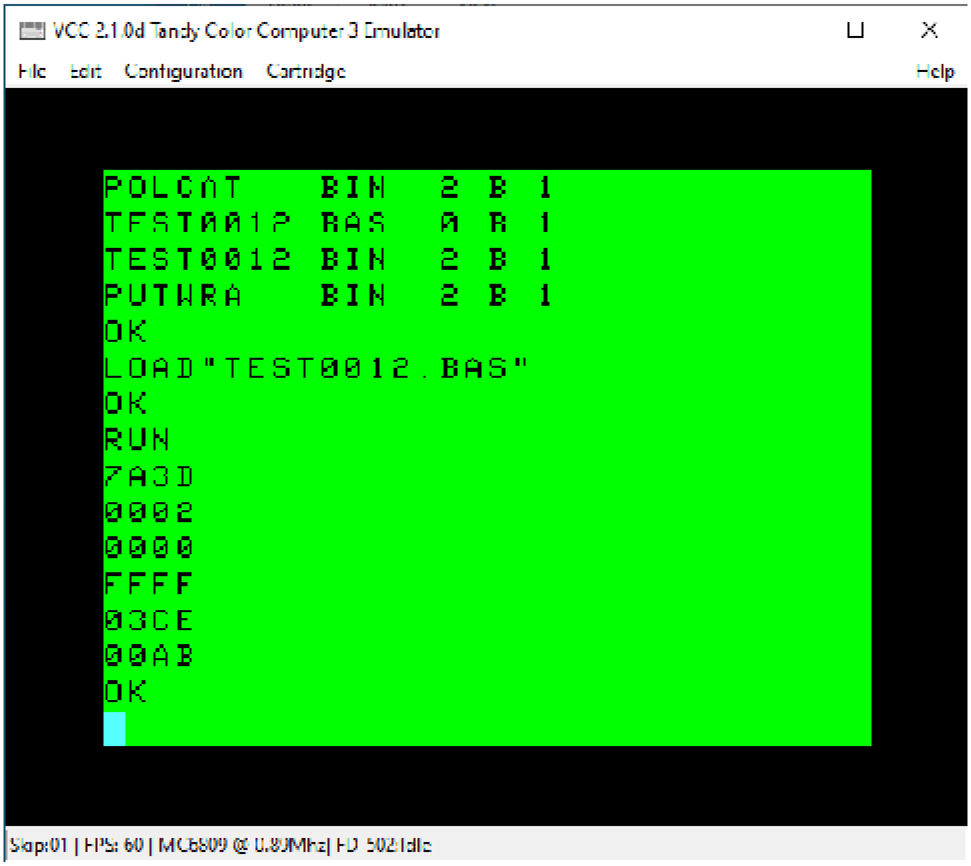
```
                    02000 * TEST ROUTINE
                    02010 * PROGRAM ORIGIN:
7000                02030       ORG     $7000
                    02040
7000 CC   7A3D      02050       LDD     #$7A3D
7003 17   AE00      02060       LBSR    PUTWRA
7006 17   AD83      02070       LBSR    CRLF
7009 CC   0002      02080       LDD     #$02
700C 17   ADF7      02090       LBSR    PUTWRA
700F 17   AD7A      02100       LBSR    CRLF
7012 CC   0000      02110       LDD     #$0
7015 17   ADEE      02120       LBSR    PUTWRA
7018 17   AD71      02130       LBSR    CRLF
701B CC   FFFF      02140       LDD     #$FFFF
701E 17   ADE5      02150       LBSR    PUTWRA
7021 17   AD68      02160       LBSR    CRLF
7024 CC   03CE      02170       LDD     #$3CE
7027 17   ADDC      02180       LBSR    PUTWRA
702A 17   AD5F      02190       LBSR    CRLF
702D CC   00AB      02200       LDD     #$00AB
7030 17   ADD3      02210       LBSR    PUTWRA
7033 17   AD56      02220       LBSR    CRLF
                    32000
                    32010 * EXIT
7036 39             32020       RTS
          0000      32767       END

00000 TOTAL ERRORS
```

The BASIC Language Control Program:

```
1000 '*****
1010 '*
1020 '* TEST0012.BAS
1030 '* MDJ 2021/09/09
1040 '*
1050 '* TEST OF
1060 '* PUTWRA.ASM
1070 '*
1080 '*****
1090 '
1100 'SETUP MEMORY
1110 PCLEAR 1
1120 CLEAR 200, &H1C00
1130 '
1140 'LOAD ML FOUNDATION
1150 'CORE ROUTINES
```

```
1160 LOADM "REGXFR.BIN"
1170 LOADM "VIDCLS.BIN"
1180 LOADM "PUTCHR.BIN"
1190 LOADM "GETCHR.BIN"
1200 LOADM "PUTBYT.BIN"
1210 LOADM "SCROLL.BIN"
1220 LOADM "PUTCHA.BIN"
1230 LOADM "PUTBYA.BIN"
1240 LOADM "CRLF.BIN"
1250 LOADM "PK2PRT.BIN"
1260 LOADM "PRT2PK.BIN"
1270 LOADM "POLCAT.BIN"
1280 '
2000 'LOAD THE ROUTINE
2010 'BEING DEVELOPED
2020 LOADM "PUTWRA.BIN"
2030 '
2040 'LOAD THE TEST ROUTINE
2050 LOADM "TEST0012.BIN"
2060 '
2070 'GO DO THE TEST
2080 EXEC &H7000
2090 '
32767 END
```

Result:



As expected.

=====

PUTWRD: Put a 16-bit Number To the VIDRAM Screen As Four Hexadecimal Digits At a Specific Position

In the same way that an 8-bit number is called a byte, a 16-bit number is called a word. In this section, the PUTWRD Routine is presented. The mnemonic PUTWRD simply stands for “Put Word”. In the previous section, PUTWRA stood for “Put Word with Advance”.

```
      *****
      *
      * PUTWRD.ASM
      * MDJ 2021/09/10
      *
      * PUTS A 16-BIT NUMBER
      * TO VIDRAM AS FOUR
      * HEXADECIMAL DIGITS.
      *
      * ENTRY CONDITIONS:
      *   D = THE 16-BIT NUMBER
      *   X = SCREEN LOCATION
      *        ($0400 - $05FC)
      *        CANNOT BE MORE
      *        THAN $05FC
      *        BECAUSE NEED
      *        ROOM TO PUT
      *        4 CHARACTERS
      *
      * EXIT CONDITIONS:
      *   X = NEW SCREEN LOC
      *        ($0404 - $0600)
      *        $0600 INDICATES
      *        END OF VIDRAM
      *        HAS BEEN PASSED
      *
      *****

      * LOW RAM CURSOR ADDRESS
0088  CURPOS  EQU     $0088

      * SCREEN ADDRESSES
      * START OF VIDRAM
```

```
          0400      00430 VIDRAM  EQU     $0400
                    00440
                    00450 * ONE BYTE PAST THE
                    00460 * END OF VIDRAM
          0600      00470 VIDEND  EQU     $0600
                    00480
                    00490 * ML FOUNDATION
                    00500 * CORE ADDRESSES
          1C00      00510 REGXFR  EQU     $1C00
          1C0E      00520 VIDCLS  EQU     $1C0E
          1C1F      00530 PUTCHR  EQU     $1C1F
          1CD2      00540 GETCHR  EQU     $1CD2
          1CD5      00550 PUTBYT  EQU     $1CD5
          1D17      00560 SCROLL  EQU     $1D17
          1D36      00570 PUTCHA  EQU     $1D36
          1D57      00580 PUTBYA  EQU     $1D57
          1D8C      00590 CRLF    EQU     $1D8C
          1DBD      00600 PK2PRT  EQU     $1DBD
          1DDA      00610 PRT2PK  EQU     $1DDA
          1DF7      00620 POLCAT  EQU     $1DF7
                    00630
                    00640 * FOR REFERENCE ONLY:
                    00650 * ONE BYTE PAST THE
                    00660 * END OF THE
                    00670 * ML FOUNDATION CORE
          1E06      00680 COREND  EQU     $1E06
                    00690
                    00700 * PROGRAM BEING DEVELOPED
                    00710 * PROGRAM ORIGIN:
                    00720 * (CHANGE AS NEEDED)
1E06                00730         ORG     $1E06
                    00740
                    00750 * SAVE THE LOW BYTE
1E06 34   04        00760 PUTWRD  PSHS    B
                    00770
                    00780 * PRINT THE HIGH BYTE
1E08 17   FECA      00790         LBSR    PUTBYT
                    00800
                    00810 * RESTORE THE LOW BYTE
                    00820 * BUT TO REGISTER A
1E0B 35   02        00830         PULS    A
                    00840
                    00850 * PRINT THE LOW BYTR
1E0D 17   FEC5      00860         LBSR    PUTBYT
                    00870
                    00880 * EXIT
1E10 39             00890         RTS
```

```
        0000     32767          END

00000 TOTAL ERRORS
```

The Assembly Language Test Routine:

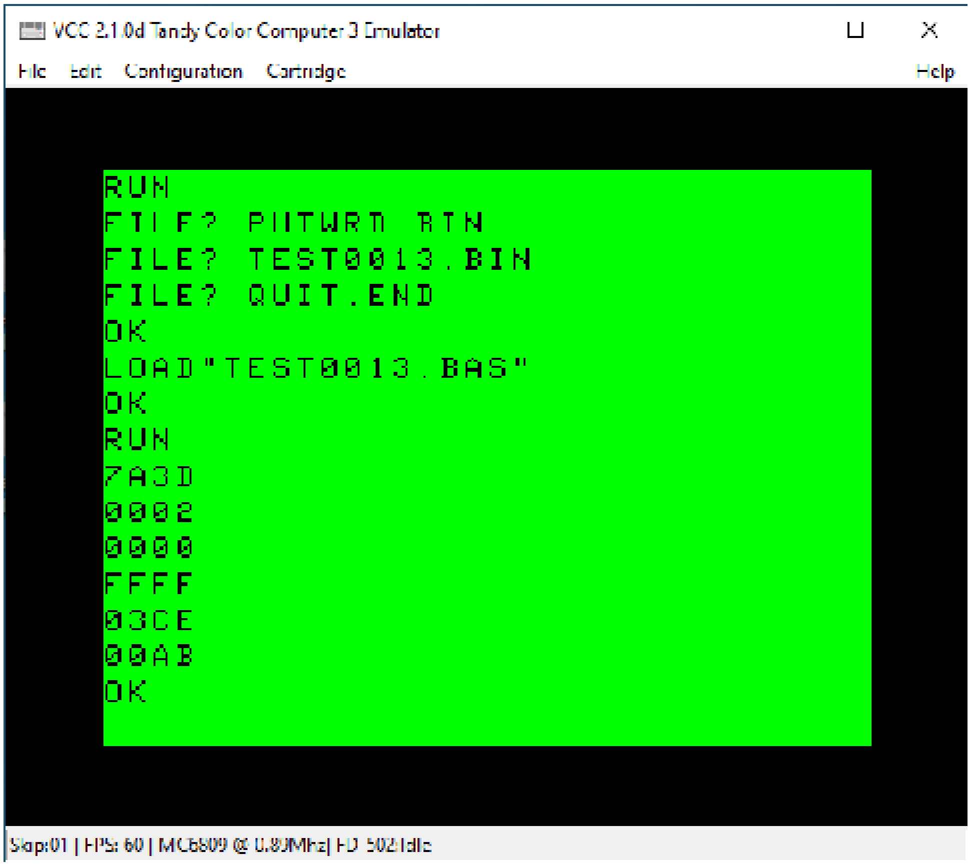
```
                 00100 *****
                 00110 *
                 00120 * TEST0013.ASM
                 00130 * MDJ 2021/09/11
                 00140 *
                 00150 * TEST OF
                 00160 * PUTWRD.ASM
                 00170
                 01000 *
                 01010 *****
                 01020
                 01030 * LOW RAM CURSOR ADDRESS
        0088     01040 CURPOS  EQU     $0088
                 01050
                 01060 * SCREEN ADDRESSES
                 01070 * START OF VIDRAM
        0400     01080 VIDRAM  EQU     $0400
                 01090
                 01100 * ONE BYTE PAST THE
                 01110 * END OF VIDRAM
        0600     01120 VIDEND  EQU     $0600
                 01130
                 01140 * ML FOUNDATION
                 01150 * CORE ADDRESSES
        1C00     01160 REGXFR  EQU     $1C00
        1C0E     01170 VIDCLS  EQU     $1C0E
        1C1F     01180 PUTCHR  EQU     $1C1F
        1CD2     01190 GETCHR  EQU     $1CD2
        1CD5     01200 PUTBYT  EQU     $1CD5
        1D17     01210 SCROLL  EQU     $1D17
        1D36     01220 PUTCHA  EQU     $1D36
        1D57     01230 PUTBYA  EQU     $1D57
        1D8C     01240 CRLF    EQU     $1D8C
        1DBD     01250 PK2PRT  EQU     $1DBD
        1DDA     01260 PRT2PK  EQU     $1DDA
        1DF7     01270 POLCAT  EQU     $1DF7
                 01280
                 01290 * FOR REFERENCE ONLY:
                 01300 * ONE BYTE PAST THE
```

```
                 01310 * END OF THE
                 01320 * ML FOUNDATION CORE
          1E06   01330 COREND  EQU     $1E06
                 01340
                 01800 * PROGRAM BEING DEVELOPED
                 01810 * EQUATE - CHANGE AS
                 01820 * NEEDED
          1E06   01830 PUTWRD  EQU     $1E06
                 01840
                 02000 * TEST ROUTINE
                 02010 * PROGRAM ORIGIN:
                 02020 * (CHANGE AS NEEDED)
7000             02030         ORG     $7000
                 02040
7000 CC   7A3D   02050         LDD     #$7A3D
7003 8D   2D     02060         BSR     L0001
7005 17   AD84   02070         LBSR    CRLF
7008 CC   0002   02080         LDD     #$02
700B 8D   25     02090         BSR     L0001
700D 17   AD7C   02100         LBSR    CRLF
7010 CC   0000   02110         LDD     #$0
7013 8D   1D     02120         BSR     L0001
7015 17   AD74   02130         LBSR    CRLF
7018 CC   FFFF   02140         LDD     #$FFFF
701B 8D   15     02150         BSR     L0001
701D 17   AD6C   02160         LBSR    CRLF
7020 CC   03CE   02170         LDD     #$3CE
7023 8D   0D     02180         BSR     L0001
7025 17   AD64   02190         LBSR    CRLF
7028 CC   00AB   02200         LDD     #$00AB
702B 8D   05     02210         BSR     L0001
702D 17   AD5C   02220         LBSR    CRLF
7030 20   07     02230         BRA     L0002
                 02240
7032 9E   88     02250 L0001   LDX     CURPOS
7034 17   ADCF   02260         LBSR    PUTWRD
7037 9F   88     02270         STX     CURPOS
                 02280
7039 12          02290 L0002   NOP
                 32000
                 32010 * EXIT
703A 39          32020         RTS
          0000   32767         END

00000 TOTAL ERRORS
```

The BASIC Language Control Program:

```
1000 '*****
1010 '*
1020 '* TEST0013.BAS
1030 '* MDJ 2021/09/11
1040 '*
1050 '* TEST OF
1060 '* PUTWRD.ASM
1070 '*
1080 '*****
1090 '
1100 'SETUP MEMORY
1110 PCLEAR 1
1120 CLEAR 200, &H1C00
1130 '
1140 'LOAD ML FOUNDATION
1150 'CORE ROUTINES
1160 LOADM "REGXFR.BIN"
1170 LOADM "VIDCLS.BIN"
1180 LOADM "PUTCHR.BIN"
1190 LOADM "GETCHR.BIN"
1200 LOADM "PUTBYT.BIN"
1210 LOADM "SCROLL.BIN"
1220 LOADM "PUTCHA.BIN"
1230 LOADM "PUTBYA.BIN"
1240 LOADM "CRLF.BIN"
1250 LOADM "PK2PRT.BIN"
1260 LOADM "PRT2PK.BIN"
1270 LOADM "POLCAT.BIN"
1280 '
2000 'LOAD THE ROUTINE
2010 'BEING DEVELOPED
2020 LOADM "PUTWRD.BIN"
2030 '
2040 'LOAD THE TEST ROUTINE
2050 LOADM "TEST0013.BIN"
2060 '
2070 'GO DO THE TEST
2080 EXEC &H7000
2090 '
32767 END
```

Result:



As expected.

=====

BKSPCE: Do a Backspace on the VIDRAM Screen

As noted in this Routine's opening comments, it backs the cursor up one position and overwrites that position with a blank space.

```
               *****
               *
               * BKSPCE.ASM
               * MDJ 2021/09/10
               *
               * BACKS THE CURSOR UP ONE
               * POSITION IF IT IS NOT
               * ALREADY AT LOWEST
               * POSITION ON THE VIDRAM
               * SCREEN AND OVERWRITES
               * THAT POSITION WITH A
               * SPACE (POKE MECHANISM
               * CODE POINT 096)
               *
               * ENTRY CONDITIONS:
               *   NONE
               *
               * EXIT CONDITIONS:
               *   NONE
               *
               *****

               * LOW RAM CURSOR ADDRESS
0088           CURPOS  EQU     $0088

               * SCREEN ADDRESSES
               * START OF VIDRAM
0400           VIDRAM  EQU     $0400

               * ONE BYTE PAST THE
               * END OF VIDRAM
0600           VIDEND  EQU     $0600

               * ML FOUNDATION
               * CORE ADDRESSES
1C00           REGXFR  EQU     $1C00
1C0E           VIDCLS  EQU     $1C0E
1C1F           PUTCHR  EQU     $1C1F
1CD2           GETCHR  EQU     $1CD2
```

```
          1CD5     00490 PUTBYT  EQU     $1CD5
          1D17     00500 SCROLL  EQU     $1D17
          1D36     00510 PUTCHA  EQU     $1D36
          1D57     00520 PUTBYA  EQU     $1D57
          1D8C     00530 CRLF    EQU     $1D8C
          1DBD     00540 PK2PRT  EQU     $1DBD
          1DDA     00550 PRT2PK  EQU     $1DDA
          1DF7     00560 POLCAT  EQU     $1DF7
                   00570
                   00580 * FOR REFERENCE ONLY:
                   00590 * ONE BYTE PAST THE
                   00600 * END OF THE
                   00610 * ML FOUNDATION CORE
          1E06     00620 COREND  EQU     $1E06
                   00630
                   00640 * PROGRAM BEING DEVELOPED
                   00650 * PROGRAM ORIGIN:
                   00660 * (CHANGE AS NEEDED)
1E06               00670         ORG     $1E06
                   00680
1E06 34   12       00690 BKSPCE  PSHS    A,X
                   00700
                   00710 * ADJUST THE CURSOR
1E08 9E   88       00720         LDX     CURPOS
                   00730
                   00740 * IS IT ALREADY AT START
                   00750 * OF VIDRAM SCREEN
1E0A 8C   0400     00760         CMPX    #VIDRAM
                   00770
                   00780 * GO IF NO
1E0D 22   03       00790         BHI     L0001
1E0F 8E   0401     00800         LDX     #VIDRAM+1
1E12 30   1F       00810 L0001   LEAX    -1,X
1E14 9F   88       00820         STX     CURPOS
                   00830
                   00840 * CLEAR SCREEN POSITION
1E16 86   60       00850         LDA     #96
1E18 17   FE04     00860         LBSR    PUTCHR
                   00870
                   00880 * EXIT
1E1B 35   12       00890         PULS    A,X
1E1D 39            00900         RTS
          0000     32767         END

00000 TOTAL ERRORS
```

For **BKSPCE.ASM**, both the Test Routine and the Control Program are just very simple modifications of those used for testing **POLCAT.ASM**.

The Assembly Language Test Routine:

```
                    00100 *****
                    00110 *
                    00120 * TEST0014.ASM
                    00130 * MDJ 2021/09/11
                    00140 *
                    00150 * BKSPCE TEST
                    00160 *
                    00170 *****
                    00180
                    00190 * LOW RAM CURSOR ADDRESS
          0088      00200 CURPOS  EQU     $0088
                    00210
                    00220 * EXTERNAL ROUTINE
                    00230 * ADDRESSES
          1C1F      00240 PUTCHR  EQU     $1C1F
          1D36      00250 PUTCHA  EQU     $1D36
          1D8C      00260 CRLF    EQU     $1D8C
          1DDA      00270 PRT2PK  EQU     $1DDA
          1DF7      00280 POLCAT  EQU     $1DF7
          1E06      00285 BKSPCE  EQU     $1E06
                    00290
7000                00300         ORG     $7000
                    00310
7000 34   03        00320         PSHS    A,CC
                    00330
                    00340 * DISPLAY PROMPT
7002 86   7E        00350         LDA     #126
7004 17   AD2F      00360         LBSR    PUTCHA
7007 86   60        00370         LDA     #96
7009 17   AD2A      00380         LBSR    PUTCHA
                    00390
                    00400 * GO CHECK ROM
                    00410 * FOR KEY PRESS
700C 17   ADE8      00420 L700C   LBSR    POLCAT
                    00430
                    00440 * GO IF NO KEY PRESS
700F 27   FB        00450         BEQ     L700C
                    00460
                    00470 * WAS IT THE BREAK KEY?
7011 81   03        00480         CMPA    #3
                    00490
```

```
                    00500 * GO IF YES
7013 27   26        00510        BEQ     L7047
                    00520
                    00530 * WAS IT A BACKSPACE?
7015 81   08        00540        CMPA    #8
                    00550
                    00560 * GO IF YES
7017 27   1D        00570        BEQ     L7036
                    00580
                    00590 * WAS IT A CARRIAGE
                    00600 * RETURN?
7019 81   0D        00610        CMPA    #13
                    00620
                    00630 * GO IF YES
701B 27   14        00640        BEQ     L7031
                    00650
                    00660 * WAS IT <= 031 ?
701D 81   20        00670        CMPA    #32
                    00680
                    00690 * GO IF YES (IGNORE)
701F 25   EB        00700        BLO     L700C
                    00710
                    00720 * WAS IT = 96 ?
7021 81   60        00730        CMPA    #96
                    00740
                    00750 * GO IF YES (IGNORE)
7023 27   E7        00760        BEQ     L700C
                    00770
                    00780 * WAS IT >= 123 ?
7025 81   7B        00790        CMPA    #123
                    00800
                    00810 * GO IF YES (IGNORE)
7027 24   E3        00820        BHS     L700C
                    00830
                    00840 * PUT IT TO VIDRAM
7029 17   ADAE      00850        LBSR    PRT2PK
702C 17   AD07      00860        LBSR    PUTCHA
702F 20   DB        00870        BRA     L700C
                    00880
                    00890 * DO CRLF
7031 17   AD58      00900 L7031  LBSR    CRLF
7034 20   D6        00910        BRA     L700C
                    00920
                    00930 * DO BACKSPACE
7036 17   ADCD      00940 L7036  LBSR    BKSPCE
7039 20   D1        01010        BRA     L700C
                    01020
```

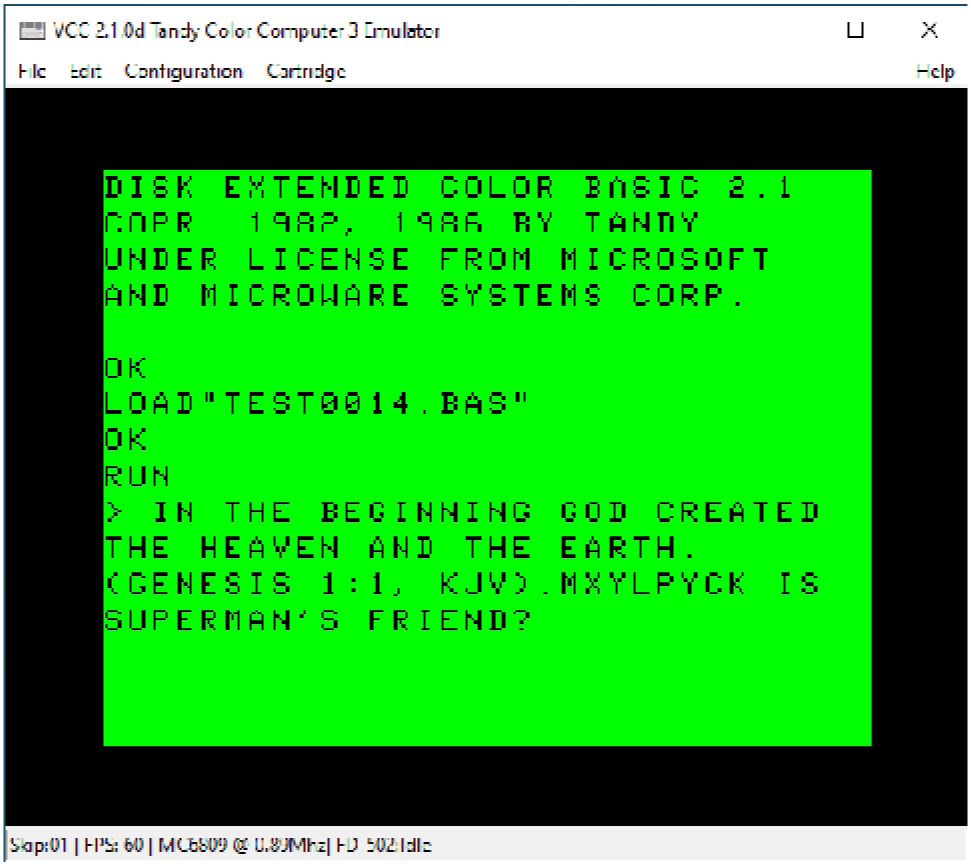
```
                 01030 * EXIT
703B 35   03     01040 L7047   PULS    A,CC
703D 39          01050         RTS
          0000   32767         END

00000 TOTAL ERRORS
```

The BASIC Language Control Program:

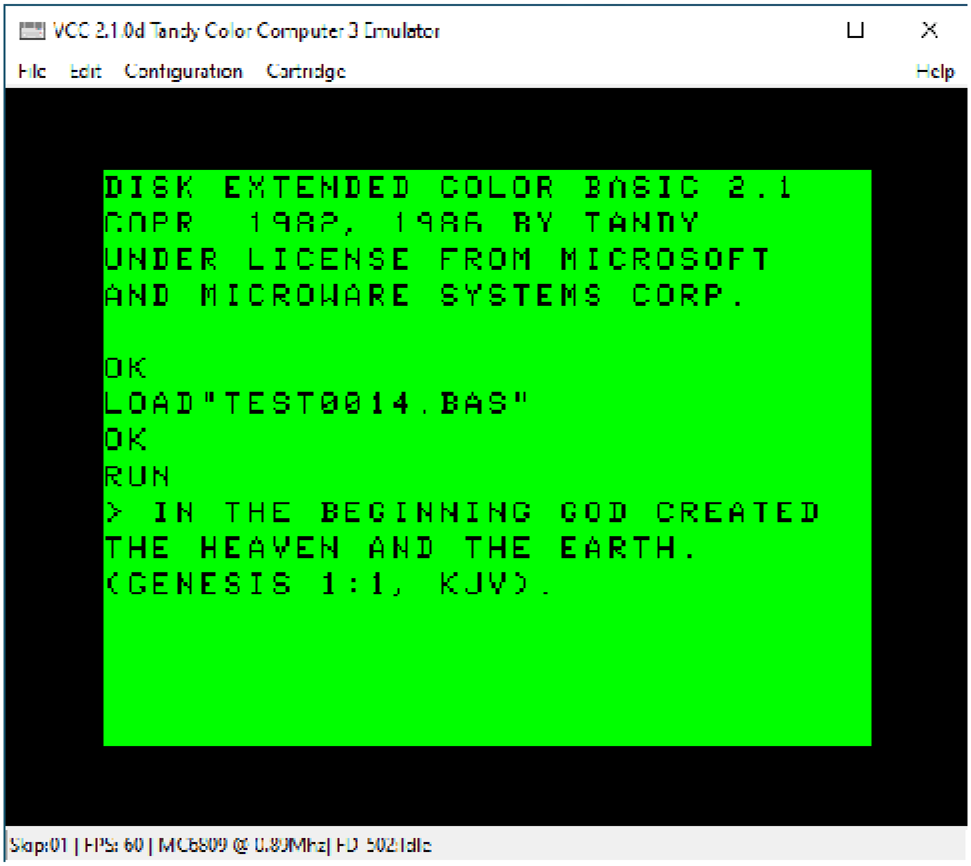
```
1000 '*****
1010 '*
1020 '* TEST0014.BAS
1030 '* MDJ 2021/09/11
1040 '*
1050 '* BKSPCE TEST
1060 '*
1070 '*****
1080 '
1090 'SETUP MEMORY
1100 PCLEAR 1
1110 CLEAR 200, &H1C00
1120 '
1130 'LOAD ML ROUTINES
1140 LOADM "REGXFR.BIN"
1150 LOADM "VIDCLS.BIN"
1160 LOADM "PUTCHR.BIN"
1170 LOADM "GETCHR.BIN"
1180 LOADM "PUTBYT.BIN"
1190 LOADM "SCROLL.BIN"
1200 LOADM "PUTCHA.BIN"
1210 LOADM "PUTBYA.BIN"
1220 LOADM "CRLF.BIN"
1230 LOADM "PK2PRT.BIN"
1240 LOADM "PRT2PK.BIN"
1250 LOADM "POLCAT.BIN"
1255 LOADM "BKSPCE.BIN"
1260 LOADM "TEST0014.BIN"
1270 '
1280 'GO DO THE TEST
1290 EXEC &H7000
1300 '
32767 END
```

Results -

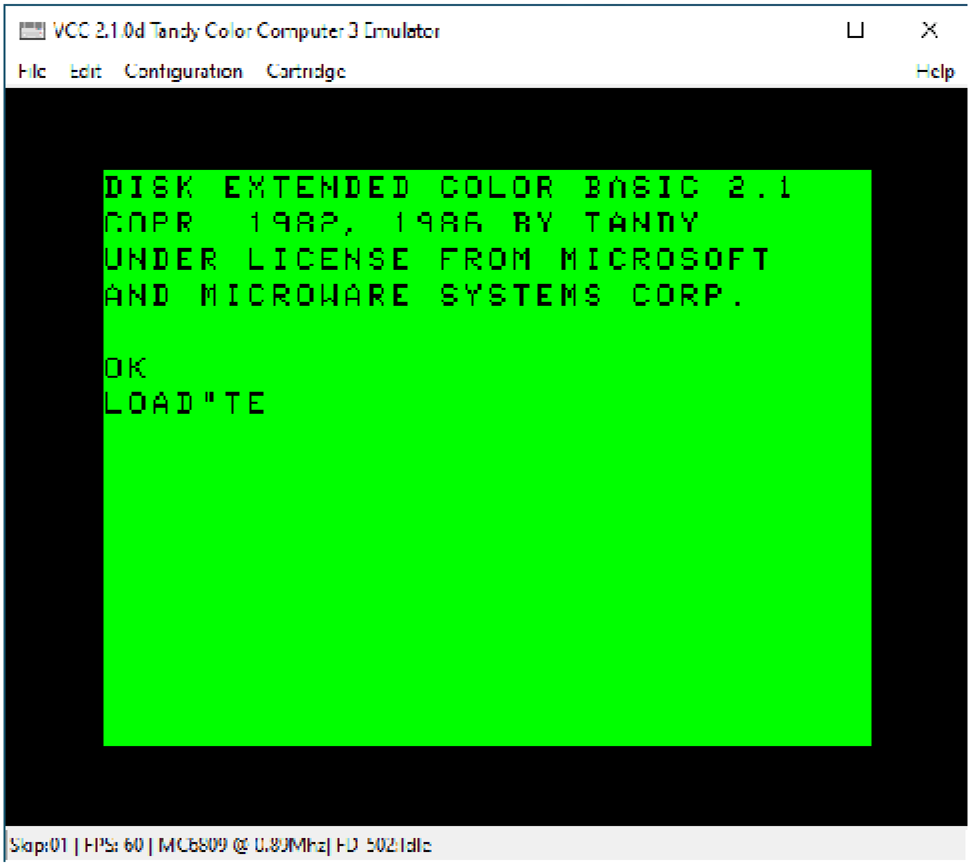
First, we start the test and type some characters:



Then, we backspace to remove some of the characters which we typed:

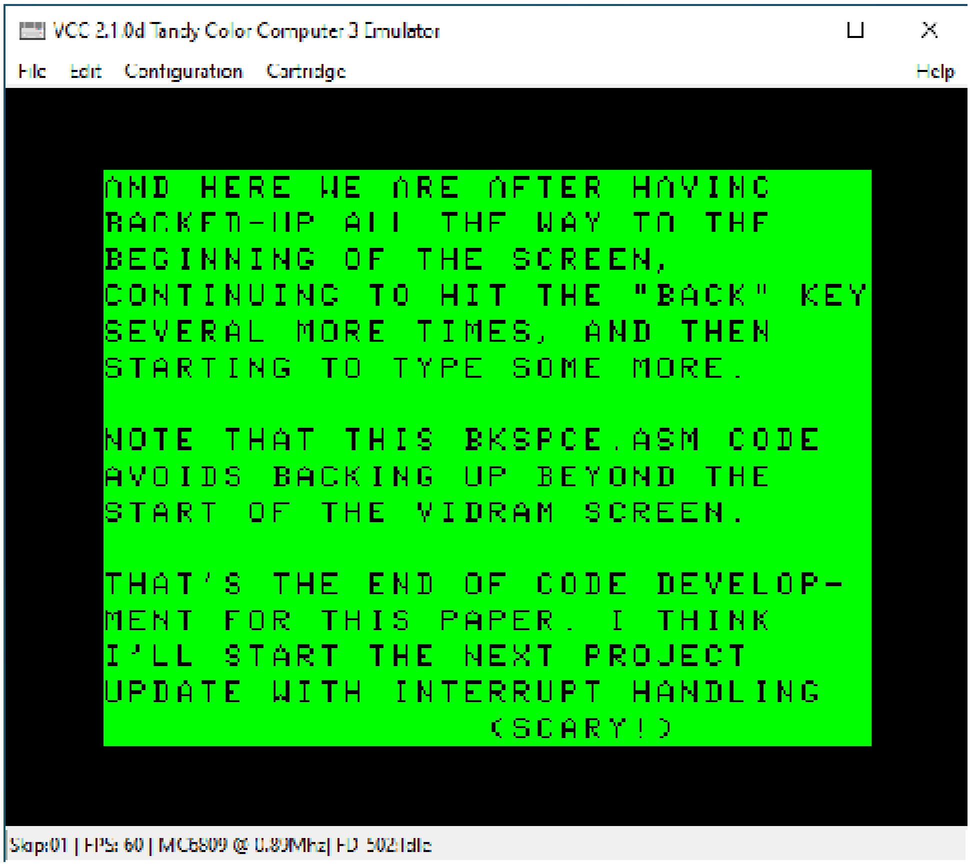


Then, we continue backing up — right past the opening prompt, the RUN command, and more:



And, we can keep on backing up, right on back to the very beginning of the VIDRAM Screen:

And then we can start all over typing again:



All this is fully as expected.

=====

GS1TST: Graphics Screen Page 1 Integrity Test

Earlier in this paper (See the ML Foundation Development Memory Map), I indicated that stuff placed in the memory originally allocated by the system (out of the box) to the Graphics Screen Page 1 was inexplicably becoming corrupted. Thus, I'm currently leaving those 1536 bytes completely unused.

I **HATE** wasting memory !!

So I've reserved a little bit of space here at the end of this paper to begin an investigation into what's happening and why. I haven't gotten very far yet, but I think I've at least begun to define the problem.

The idea behind the following test code is simply to:

1. Define an internal 1536-byte Buffer (**`TSTS`** to **`TSTE`**) and load it with sample data.
2. Copy the buffer data to Graphics Screen Page 1 memory (**`GS1S`** to **`GS1E`**).
3. Compare the Buffer and the Page to discover any data mismatches. (There shouldn't be any unless something external is interfering with the Page).

The concept included running several different sets of data through this Test Routine to see if any data dependencies were present. The data sets, themselves, were simply a series of 1536 bytes, running in byte order, e.g. … $47, $48, $49, $4A, … etc.

The sets were different from each other simply on the basis of where each series began, I ran six sets of data, beginning with bytes $00; $25; $2C; $AD; $E5; and $FF respectively. The data sets each simply rolled over where necessary, e.g. … $FE, $FF, $00, $01 … etc.

The actual set selection for a given run was made in the **`GS1TST.BAS`** Control Program, instead of in the Assembly Language Test Routine. Noting the Test Routine's line:

```
2A18 86   00       00850         LDA     #0
```

I simply did the following **`POKE`** from the Control Program:

```
          1400 POKE &H2A19,&HXX
```

where **`XX`** was the hexadecimal starting value for the set.

To ensure that the code itself worked correctly, I included a second 1536-byte internal buffer (**`DUMS`** to **`DUME`**) and ran **`TSTS`** to **`TSTE`** against it, instead of against **`GS1S`** to **`GS1E`**. (See lines 810 and 820 in the following Test Routine).

To use that second buffer, I also changed lines:

```
2A32 108E 0600     01110          LDY     #GS1S
2A42 108E 0600     01350          LDY     #GS1S
```

to:

```
2A32 108E 2415     01110          LDY     #DUMS
2A42 108E 2415     01350          LDY     #DUMS
```

and saved the altered Test Routine as **GS1DUM.ASM** instead of **GS1TST.ASM**. In use, I assembled either routine to **GS1TST.BIN** so it could be run by **GS1TST.BAS** unchanged except for the starting data value **POKE**.

The expected results from each run was:

```
RUN
XX
EOR
OK
```

where "**XX**" was the starting data value and "**EOR**" meant "End-Of-Run".

In the event of a data mismatch, the result would include a line like:

```
MM 219A=8A 0986=88
```

where: **MM** ==> Mismatch
219A = Test Buffer Address
8A = Test Buffer Data Value
0986 = Graphics Page Address
88 = Graphics Page Data Value

The results are plain: There is clearly something external interfering with Graphics Screen Page 1 and it's necessary to continue to keep that memory ($0600 to $0C00) unused and untouched, at least for now.

RUN !! RUN !! Nothing is safe **!!**

Seriously, though, until I can find out what's causing this, I'll continue to leave that Page alone. I don't know: perhaps this is an Interrupts issue - This ML Foundation Core does not yet deal with interrupts at all, even though I'm sure they're happening. Interrupts are something I'll be addressing early-on in my continuing system development. Until then … … …

The Assembly Language Test Routine:

```
              00100 *****
              00110 *
              00120 * GS1TST.ASM
              00130 * MDJ 2021/09/10
              00140 *
              00150 * CHECK THE INTEGRITY OF
              00160 * GRAPHICS SCREEN PAGE 1
              00170 * FOR USE AS SCRATCHPAD
              00180 * AND/OR GENERAL STORAGE
              00190 *
              00200 * BUT CHECK THIS PROGRAM
              00210 * ITSELF BY USING DUMS
              00220 * AND DUME INSTEAD OF
              00230 * GS1S AND GS1E
              00240 *
              00250 * ASSEMBLE TO GS1TST.BIN
              00260 * IN EITHER CASE
              00270 *
              00280 *****
              00290
              00300 * LOW RAM CURSOR ADDRESS
0088          00310 CURPOS  EQU     $0088
              00320
              00330 * SCREEN ADDRESSES
              00340 * START OF VIDRAM
0400          00350 VIDRAM  EQU     $0400
              00360
              00370 * ONE BYTE PAST THE
              00380 * END OF VIDRAM
0600          00390 VIDEND  EQU     $0600
              00400
              00410 * ML FOUNDATION
              00420 * CORE ADDRESSES
1C00          00430 REGXFR  EQU     $1C00
1C0E          00440 VIDCLS  EQU     $1C0E
1C1F          00450 PUTCHR  EQU     $1C1F
1CD2          00460 GETCHR  EQU     $1CD2
1CD5          00470 PUTBYT  EQU     $1CD5
1D17          00480 SCROLL  EQU     $1D17
1D36          00490 PUTCHA  EQU     $1D36
1D57          00500 PUTBYA  EQU     $1D57
1D8C          00510 CRLF    EQU     $1D8C
1DBD          00520 PK2PRT  EQU     $1DBD
1DDA          00530 PRT2PK  EQU     $1DDA
```

```
          1DF7      00540 POLCAT  EQU     $1DF7
                    00550
                    00560 * FOR REFERENCE ONLY:
                    00570 * ONE BYTE PAST THE
                    00580 * END OF THE
                    00590 * ML FOUNDATION CORE
          1E06      00600 COREND  EQU     $1E06
                    00610
                    00620 * ADDITIONAL EQUATES HERE
                    00630 * GRAPHICS SCREEN PAGE 1
                    00640 * BOUNDARIES
          0600      00650 GS1S    EQU     $0600
          0C00      00660 GS1E    EQU     $0C00
                    00670
                    00680 * PUT WORD ROUTINE
          1E06      00690 PUTWRA  EQU     $1E06
                    00700
                    00710 * PROGRAM BEING DEVELOPED
                    00720 * PROGRAM ORIGIN:
                    00730 * (CHANGE AS NEEDED)
1E11                00740         ORG     $1E11
                    00750
1E11 16   0C02      00760         LBRA    L0001
                    00770
                    00780 * TEST DATA
1E14                00790 TSTS    RMB     1536
2414                00800 TSTE    RMB     1
2415                00810 DUMS    RMB     1536
2A15                00820 DUME    RMB     1
                    00830
2A16 34   36        00840 L0001   PSHS    A,B,X,Y
2A18 86   00        00850         LDA     #0
                    00860
                    00870 * REPORT THE STARTING
                    00880 * TEST DATA VALUE
2A1A 34   02        00890         PSHS    A
2A1C 17   F338      00900         LBSR    PUTBYA
2A1F 17   F36A      00910         LBSR    CRLF
2A22 35   02        00920         PULS    A
                    00930
                    00940 * LOAD THE TEST DATA
2A24 8E   1E14      00950         LDX     #TSTS
2A27 A7   80        00960 L0002   STA     ,X+
                    00970
                    00980 * ROLLOVER 255-->0
                    00990 * AS REQUIRED
2A29 4C             01000         INCA
```

```
2A2A 8C   2414      * DONE LOADING?
                            CMPX    #TSTE

                    * GO IF NO
2A2D 25   F8                BLO     L0002

                    * COPY THE TEST DATA TO
                    * GRAPHICS SCREEN PAGE 1
2A2F 8E   1E14              LDX     #TSTS
2A32 108E 0600              LDY     #GS1S
2A36 A6   80        L0003   LDA     ,X+
2A38 A7   A0                STA     ,Y+

                    * DONE COPYING?
2A3A 8C   2414              CMPX    #TSTE

                    * GO IF NO
2A3D 25   F7                BLO     L0003

                    * FOR GENERAL STORAGE
                    * TEST ONLY - COMMENT OUT
                    * FOR SCRATCHPAD TEST
                    *
                    *       LDA     #127    ?
                    *       LBSR    PUTCHA
                    *       LBSR    POLCAT
                    *       LBSR    CRLF
                    *
                    * GOES OUT TO ROM AND
                    * BACK TO RAM AGAIN

                    * CHECK FOR MISMATCHES
2A3F 8E   1E14              LDX     #TSTS
2A42 108E 0600              LDY     #GS1S
2A46 A6   80        L0004   LDA     ,X+
2A48 E6   A4                LDB     ,Y

                    *MISMATCH?
2A4A A1   A0                CMPA    ,Y+

                    * GO IF YES
2A4C 26   07                BNE     L0005

                    * DONE CHECKING?
2A4E 8C   2414              CMPX    #TSTE

```

```
                  01480 * GO IF NO
2A51 25   F3      01490         BLO     L0004
                  01500
                  01510 * GO REPORT TEST COMPLETE
2A53 20   41      01520         BRA     L0006
                  01530
                  01540 * REPORT MISMATCH
2A55 34   06      01550 L0005   PSHS    A,B
2A57 86   4D      01560         LDA     #77     M
2A59 17   F2DA    01570         LBSR    PUTCHA
2A5C 86   4D      01580         LDA     #77     M
2A5E 17   F2D5    01590         LBSR    PUTCHA
2A61 86   60      01600         LDA     #96     SP
2A63 17   F2D0    01610         LBSR    PUTCHA
                  01620
                  01630 * TEST DATA ADDRESS
2A66 1F   10      01640         TFR     X,D
2A68 17   F39B    01650         LBSR    PUTWRA
2A6B 86   7D      01660         LDA     #125    =
2A6D 17   F2C6    01670         LBSR    PUTCHA
2A70 35   06      01680         PULS    A,B
2A72 34   06      01690         PSHS    A,B
                  01700
                  01710 * TEST DATA VALUE
2A74 17   F2E0    01720         LBSR PUTBYA
2A77 86   60      01730         LDA     #96     SP
2A79 17   F2BA    01740         LBSR    PUTCHA
                  01750
                  01760 * GRAPHICS ADDRESS
2A7C 1F   20      01770         TFR     Y,D
2A7E 17   F385    01780         LBSR    PUTWRA
2A81 86   7D      01790         LDA     #125    =
2A83 17   F2B0    01800         LBSR    PUTCHA
2A86 35   06      01810         PULS    A,B
2A88 34   06      01820         PSHS    A,B
2A8A 1F   98      01830         TFR     B,A
                  01840
                  01850 * GRAPHICS VALUE
2A8C 17   F2C8    01860         LBSR    PUTBYA
2A8F 17   F2FA    01870         LBSR    CRLF
2A92 35   06      01880         PULS    A,B
2A94 20   B0      01890         BRA     L0004
                  01900
                  01910 * REPORT COMPLETION, I.E.
                  01920 * "END-OF-RUN (EOR)"
2A96 34   02      01930 L0006   PSHS    A
2A98 86   45      01940         LDA     #69     E
```

```
2A9A 17   F299     01950         LBSR    PUTCHA
2A9D 86   4F       01960         LDA     #79     O
2A9F 17   F294     01970         LBSR    PUTCHA
2AA2 86   52       01980         LDA     #82     R
2AA4 17   F28F     01990         LBSR    PUTCHA
2AA7 17   F2E2     02000         LBSR    CRLF
2AAA 35   02       02010         PULS    A
                   02020
                   02030 * EXIT
2AAC 35   36       02040         PULS    A,B,X,Y
2AAE 39            02050         RTS
          0000     32767         END

00000 TOTAL ERRORS
```

The BASIC Language Control Program:

```
1000 '*****
1010 '*
1020 '* GS1TST.BAS
1030 '* MDJ 2021/09/10
1040 '*
1050 '* TEST OF
1060 '* GS1TST.ASM
1070 '*
1080 '*****
1090 '
1100 'SETUP MEMORY
1110 PCLEAR 1
1120 CLEAR 200, &H1C00
1130 '
1140 'LOAD ML FOUNDATION
1150 'CORE ROUTINES
1160 LOADM "REGXFR.BIN"
1170 LOADM "VIDCLS.BIN"
1180 LOADM "PUTCHR.BIN"
1190 LOADM "GETCHR.BIN"
1200 LOADM "PUTBYT.BIN"
1210 LOADM "SCROLL.BIN"
1220 LOADM "PUTCHA.BIN"
1230 LOADM "PUTBYA.BIN"
1240 LOADM "CRLF.BIN"
1250 LOADM "PK2PRT.BIN"
1260 LOADM "PRT2PK.BIN"
1270 LOADM "POLCAT.BIN"
```

```
1280 '
1290 'LOAD OTHER
1300 'NEEDED ROUTINES
1310 LOADM "PUTWRA.BIN"
1320 '
1330 'LOAD THE ROUTINE
1340 'BEING DEVELOPED
1350 LOADM "GS1TST.BIN"
1360 '
1370 'UNCOMMENT AND MODIFY THIS
1380 'TO MANUALLY RANDOMIZE THE
1390 'TEST DATA
1400 'POKE &H2A19,37
1410 '
1420 'GO DO THE TEST
1430 EXEC &H1E11
1440 '
32767 END
```

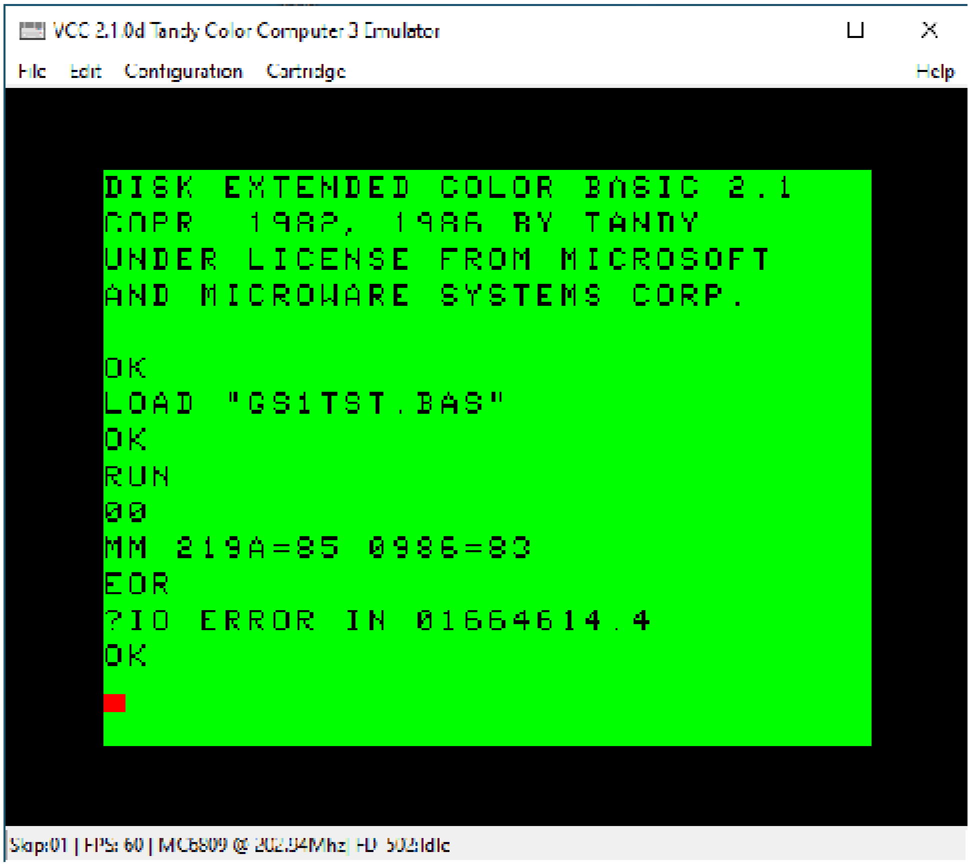
Results from **GS1DUM.ASM**:

As expected. I then ran several more tests with different starting data item values **POKE**d in from **GS1TST.BAS** and obtained the following results from those runs:

```
1400 POKE &H2A19,&H25
RUN
25
EOR
OK

1400 POKE &H2A19,&H2C
RUN
2C
EOR
OK

1400 POKE &H2A19,&HAD
RUN
AD
EOR
OK

1400 POKE &H2A19,&HE5
RUN
E5
EOR
OK

1400 POKE &H2A19,&HFF
RUN
FF
EOR
OK
```

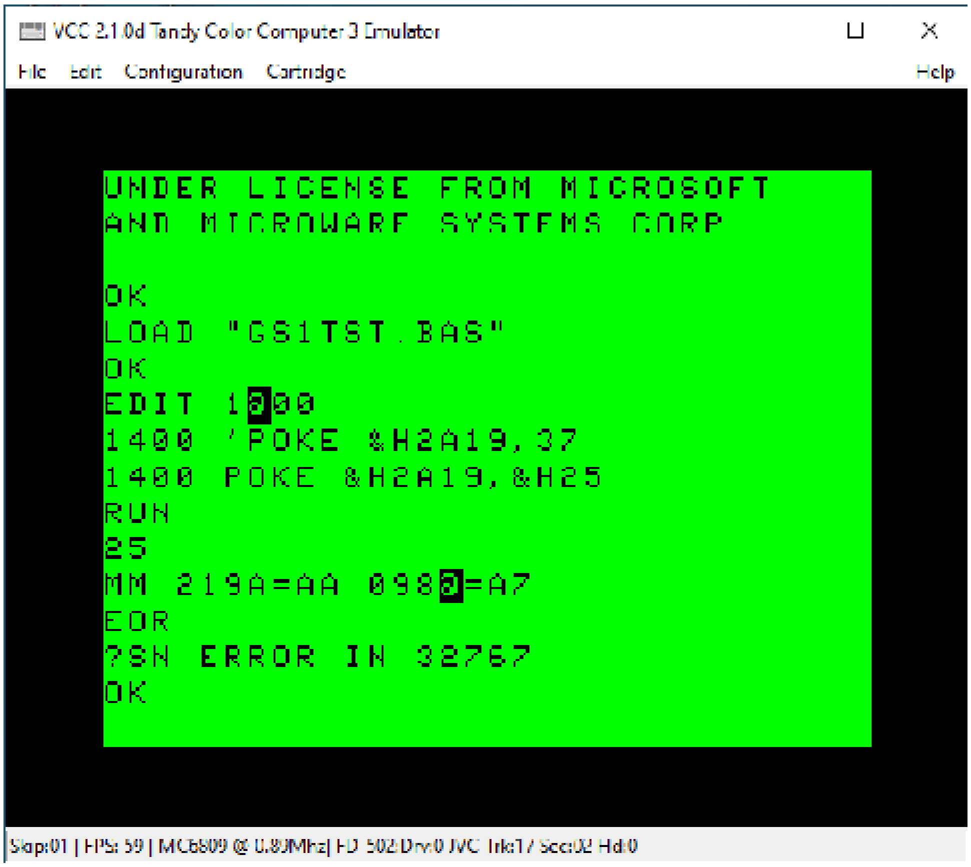
All as expected.

Results from **`GS1TST.ASM`**:



Most Definitely NOT as expected. In fact, this run also thoroughly corrupted the disk. Having already destroyed a few disks in the process of putting these tests together, I now had several backups on hand.

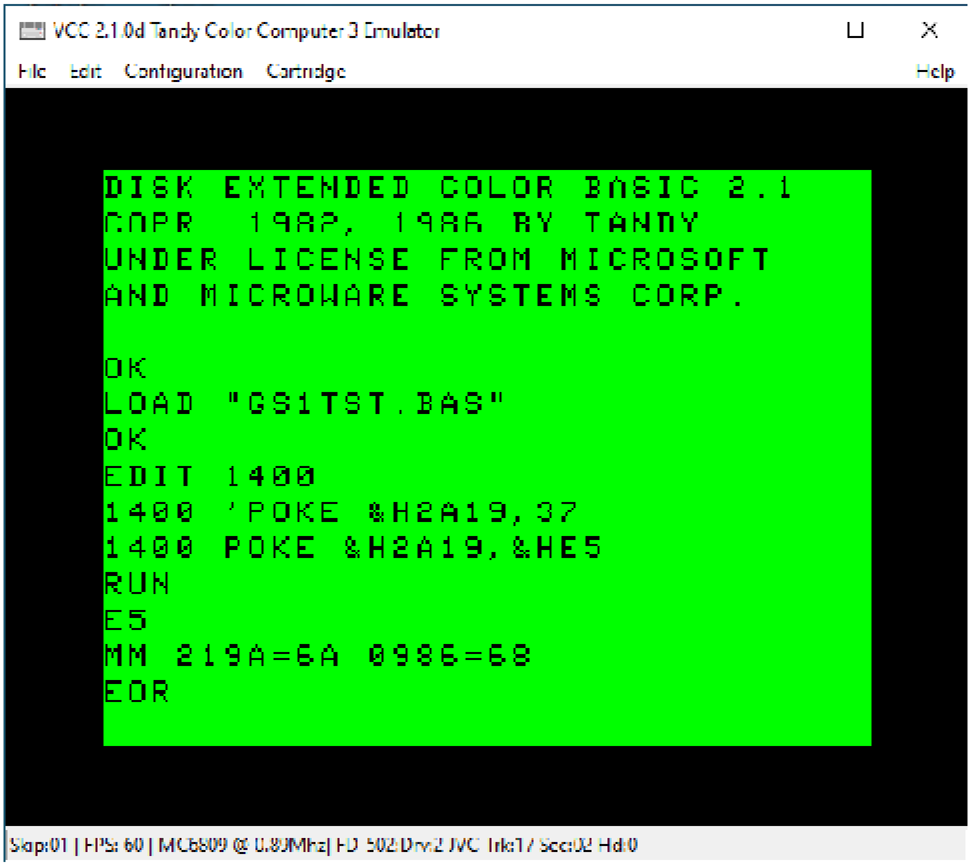
Onwards: I ran tests on **GS1TST.ASM** with the same starting data item values as in my **GS1DUM.ASM** runs above, again **POKE**d in from **GS1TST.BAS.** I obtained the following results from those runs:



Note the corruption of the screen with the two Reversed @ symbols in lines which had already been printed.


VCC 2.1.0d Tandy Color Computer 3 Emulator
File Edit Configuration Cartridge Help
DISK EXTENDED COLOR BASIC 2.1
COPR 1982, 1986 BY TANDY
UNDER LICENSE FROM MICROSOFT
AND MICROWARE SYSTEMS CORP.
OK
LOAD "GS1TST.BAS"
OK
EDIT 1400
1400 'POKE &H2A19,37
1400 POKE &H2A19,&H2C
RUN
2C
MM 219A=B1 0986=AE
EOR
Skip:01 | FPS: 60 | MC6809 @ 0.89Mhz| FD-502:Drv:0 JVC Trk:17 Sec:03 Hd:0

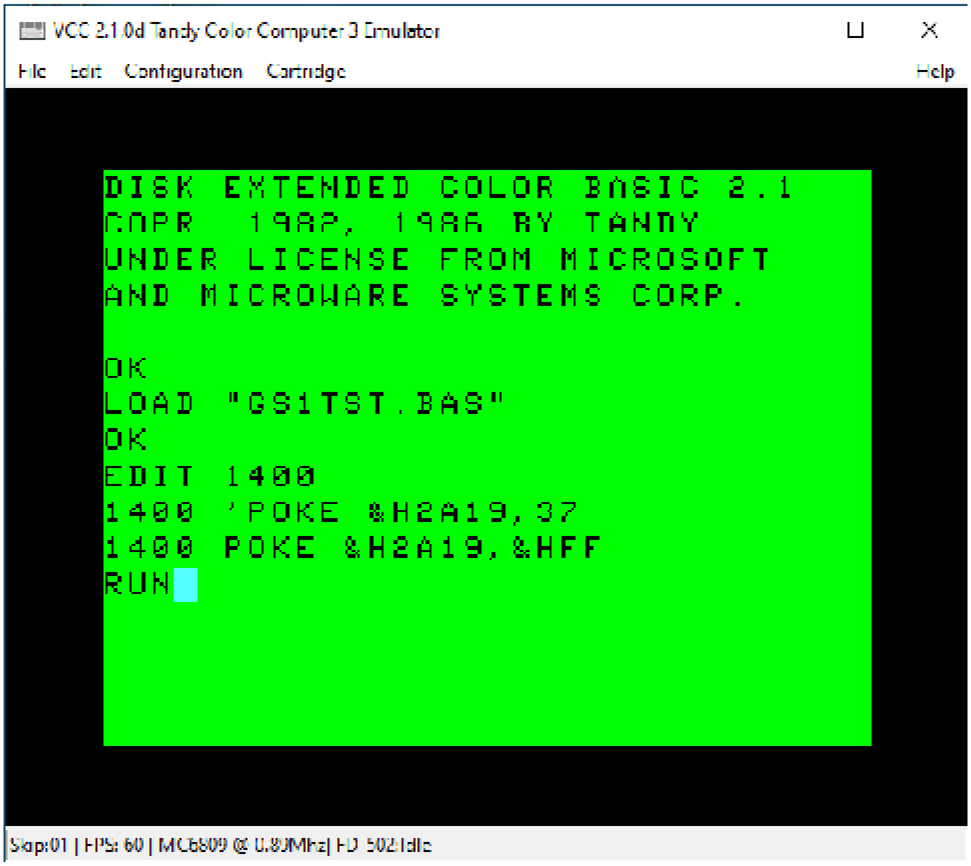
Note the “restart”. Actually, the system is corrupt at this point, and will not run without a Hard Reset. That’s generally the case with all six of these **GS1TST.ASM** tests against the Graphics Page.



In this test, the system didn't even get back to the Command Prompt.

This last test was particularly interesting in that it reported the starting address (i.e. “FF”) correctly for an instant, but then blew the whole screen and left us with the result above.

Just before hitting the “ENTER” key after “RUN” and getting the momentary “FF”, the screen looked like this:


VCC 2.1.0d Tandy Color Computer 3 Emulator
File Edit Configuration Cartridge Help
DISK EXTENDED COLOR BASIC 2.1
COPR 1982, 1986 BY TANDY
UNDER LICENSE FROM MICROSOFT
AND MICROWARE SYSTEMS CORP.
OK
LOAD "GS1TST.BAS"
OK
EDIT 1400
1400 'POKE &H2A19,37
1400 POKE &H2A19,&HFF
RUN
Skip:01 | FPS: 60 | MC6809 @ 0.89Mhz| FD-502:Idle

=====

Results

With the exception of GS1TST: Graphics Screen Page 1 Integrity Test, the development and preliminary testing of these components of the The ML Foundation Core are complete and correct.

This does not preclude the possibility that errors in these routines may be encountered during the continued development of the Core and the rest of The ML Foundation.

Nor does it preclude the possibility that you may discover some errors which I've missed. If you do, please let me know.

M.D.J. 2021/09/29
info@bds-soft.com

=====

Conclusions and Future Work

At this point, the preliminary pieces of The ML Foundation Core; what little they are intended to do (i.e. very, very minimal I/O), they do correctly.

Please take them, play with them, experiment, try to break them, etc. Let me know how you fare, if you will.

Meanwhile, I will continue with further development of The ML Foundation. Very first on the agenda will probably be Interrupts. After that, the work will split into two parallel paths which will proceed somewhat concurrently:

1. All-Assembly Language Routines where speed is a major factor, and

2. Routines that jump into ROM to handle inherently slower tasks.

Among the latter will be Assembly Language Routines to access ROM code for BASIC Language commands and functions such as **AUDIO, BACKUP, CLOADM, CLOSE, COPY, CSAVEM, DIR, DRIVE, DSKI$, DSKINI, DSKO$, EOF, FREE, GET, INPUT#, JOYSTK, KILL, LINE INPUT, LLIST, LOADM, LOC, MOTOR, OPEN, POS, PRINT#, PRINT# USING, PUT#, RENAME, SAVEM, SKIPF, UNLOAD, VERIFY OFF, VERIFY ON,** and **WRITE.**

Among the former will be Assembly Language Routines analogous to BASIC Language commands and functions such as **ABS, ASC, ATN, CHR$, CONT, COS, CVN, EXP, FIELD, FILES, HEX$, INSTR, INT, LEN, LIST, LOF, LOG, LSET, MEM, MKN$, MID$, NEW, PLAY, RIGHT$, RND, RSET, SGN, SIN, SOUND, SQR, STOP, STRING$, STR$, TAN, TIMER,** and **VAL.**

In addition to these items, and in many cases prior to them, I'll be working on code for:

1. Signed and Unsigned 32-bit and 64-bit integer math,

2. IEEE Standard 754 32-bit and 64-bit floating-point math, and

3. Advanced text and string processing.

Anything to do with graphics will wait until later. (Maybe you'll develop such Routines before I do - Give it a go!)

=====

Appendix A
Decimal to Hexadecimal Conversions

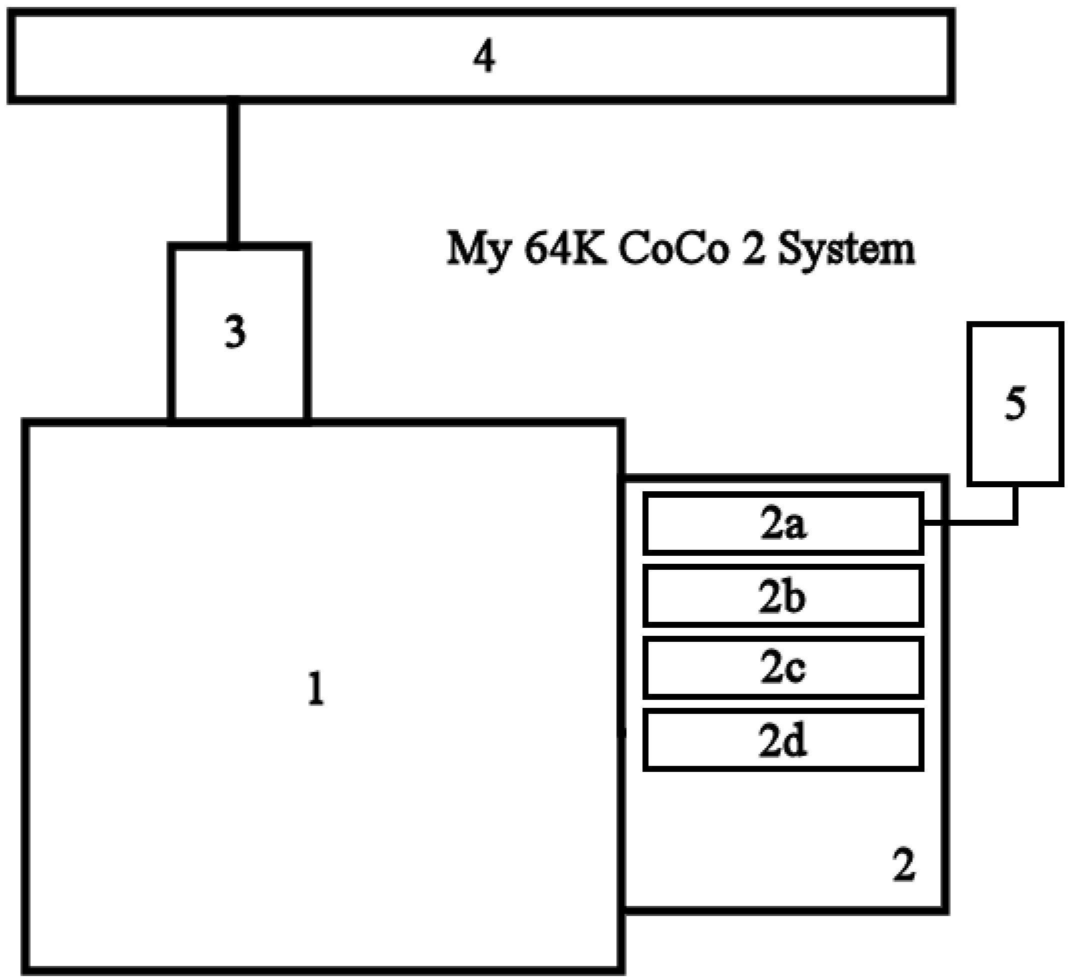
DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX
000	00	032	20	064	40	096	60
001	01	033	21	065	41	097	61
002	02	034	22	066	42	098	62
003	03	035	23	067	43	099	63
004	04	036	24	068	44	100	64
005	05	037	25	069	45	101	65
006	06	038	26	070	46	102	66
007	07	039	27	071	47	103	67
008	08	040	28	072	48	104	68
009	09	041	29	073	49	105	69
010	0A	042	2A	074	4A	106	6A
011	0B	043	2B	075	4B	107	6B
012	0C	044	2C	076	4C	108	6C
013	0D	045	2D	077	4D	109	6D
014	0E	046	2E	078	4E	110	6E
015	0F	047	2F	079	4F	111	6F
016	10	048	30	080	50	112	70
017	11	049	31	081	51	113	71
018	12	050	32	082	52	114	72
019	13	051	33	083	53	115	73
020	14	052	34	084	54	116	74
021	15	053	35	085	55	117	75
022	16	054	36	086	56	118	76
023	17	055	37	087	57	119	77
024	18	056	38	088	58	120	78
025	19	057	39	089	59	121	79
026	1A	058	3A	090	5A	122	7A
027	1B	059	3B	091	5B	123	7B
028	1C	060	3C	092	5C	124	7C
029	1D	061	3D	093	5D	125	7D
030	1E	062	3E	094	5E	126	7E
031	1F	063	3F	095	5F	127	7F

DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX
128	**80**	**160**	**A0**	**192**	**C0**	**224**	**E0**
129	**81**	**161**	**A1**	**193**	**C1**	**225**	**E1**
130	**82**	**162**	**A2**	**194**	**C2**	**226**	**E2**
131	**83**	**163**	**A3**	**195**	**C3**	**227**	**E3**
132	**84**	**164**	**A4**	**196**	**C4**	**228**	**E4**
133	**85**	**165**	**A5**	**197**	**C5**	**229**	**E5**
134	**86**	**166**	**A6**	**198**	**C6**	**230**	**E6**
135	**87**	**167**	**A7**	**199**	**C7**	**231**	**E7**
136	**88**	**168**	**A8**	**200**	**C8**	**232**	**E8**
137	**89**	**169**	**A9**	**201**	**C9**	**233**	**E9**
138	**8A**	**170**	**AA**	**202**	**CA**	**234**	**EA**
139	**8B**	**171**	**AB**	**203**	**CB**	**235**	**EB**
140	**8C**	**172**	**AC**	**204**	**CC**	**236**	**EC**
141	**8D**	**173**	**AD**	**205**	**CD**	**237**	**ED**
142	**8E**	**174**	**AE**	**206**	**CE**	**238**	**EE**
143	**8F**	**175**	**AF**	**207**	**CF**	**239**	**EF**
144	**90**	**176**	**B0**	**208**	**D0**	**240**	**F0**
145	**91**	**177**	**B1**	**209**	**D1**	**241**	**F1**
146	**92**	**178**	**B2**	**210**	**D2**	**242**	**F2**
147	**93**	**179**	**B3**	**211**	**D3**	**243**	**F3**
148	**94**	**180**	**B4**	**212**	**D4**	**244**	**F4**
149	**95**	**181**	**B5**	**213**	**D5**	**245**	**F5**
150	**96**	**182**	**B6**	**214**	**D6**	**246**	**F6**
151	**97**	**183**	**B7**	**215**	**D7**	**247**	**F7**
152	**98**	**184**	**B8**	**216**	**D8**	**248**	**F8**
153	**99**	**185**	**B9**	**217**	**D9**	**249**	**F9**
154	**9A**	**186**	**BA**	**218**	**DA**	**250**	**FA**
155	**9B**	**187**	**BB**	**219**	**DB**	**251**	**FB**
156	**9C**	**188**	**BC**	**220**	**DC**	**252**	**FC**
157	**9D**	**189**	**BD**	**221**	**DD**	**253**	**FD**
158	**9E**	**190**	**BE**	**222**	**DE**	**254**	**FE**
159	**9F**	**191**	**BF**	**223**	**DF**	**255**	**FF**

=====

Appendix B: My 64K CoCo 2 System

As of 2021/09/08, my physical 64K CoCo 2 System is depicted in the block diagram below.



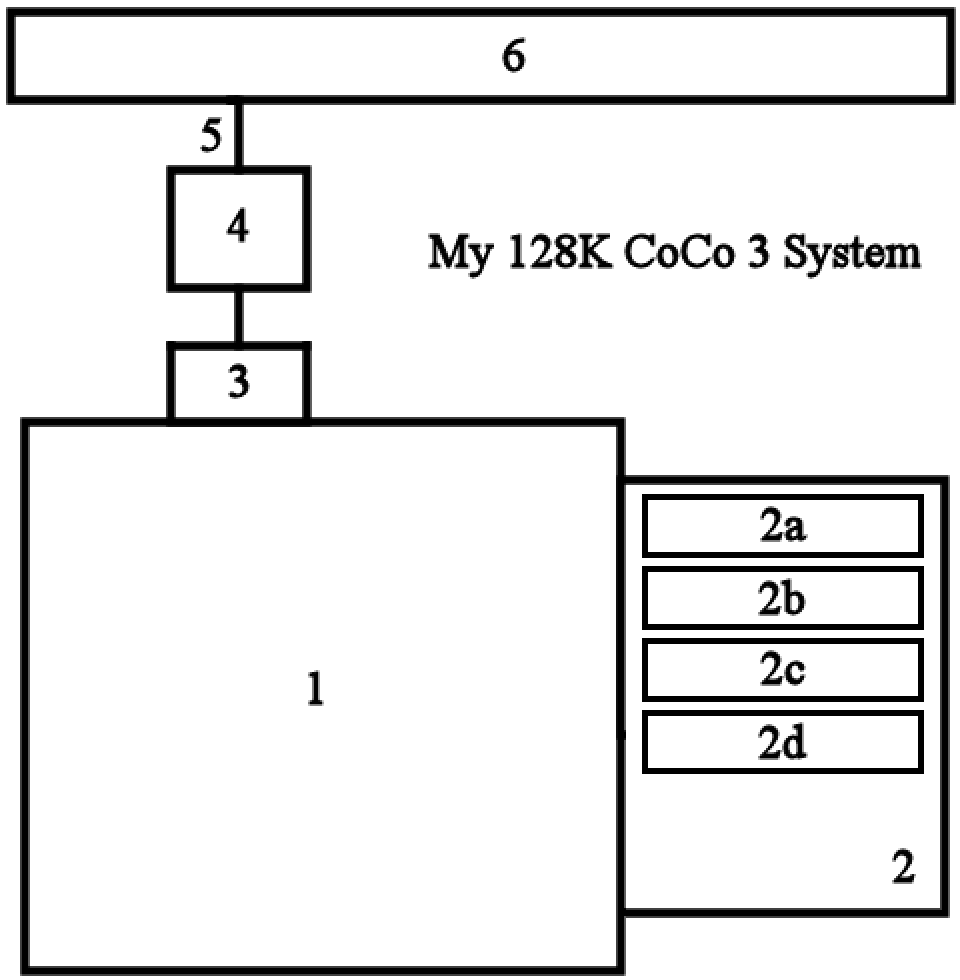
System Legend:

1. 64K CoCo 2, Model Number 26-3127, Serial Number 007601.
2. Multi-Pak Interface, Model Number 26-3124, Serial Number 2005259
    - 2a. Floppy Disk Controller, Model Number 26-3029
    - 2b. CoCo SDC, running SDC-DOS 1.6 CC2.
    - 2c. RS-232 Pak
    - 2d. Empty slot
3. CoCo VGA.
4. RCA 19” VGA/HDMI Monitor, Model RT1970.
5. 5.25” Disk Drive, Model Number 26-3022

=====

Appendix C: My 128K CoCo 3 System

As of 2021/09/08, my physical 128K CoCo 3 System is depicted in the block diagram below.



See following page for System Legend.

System Legend:

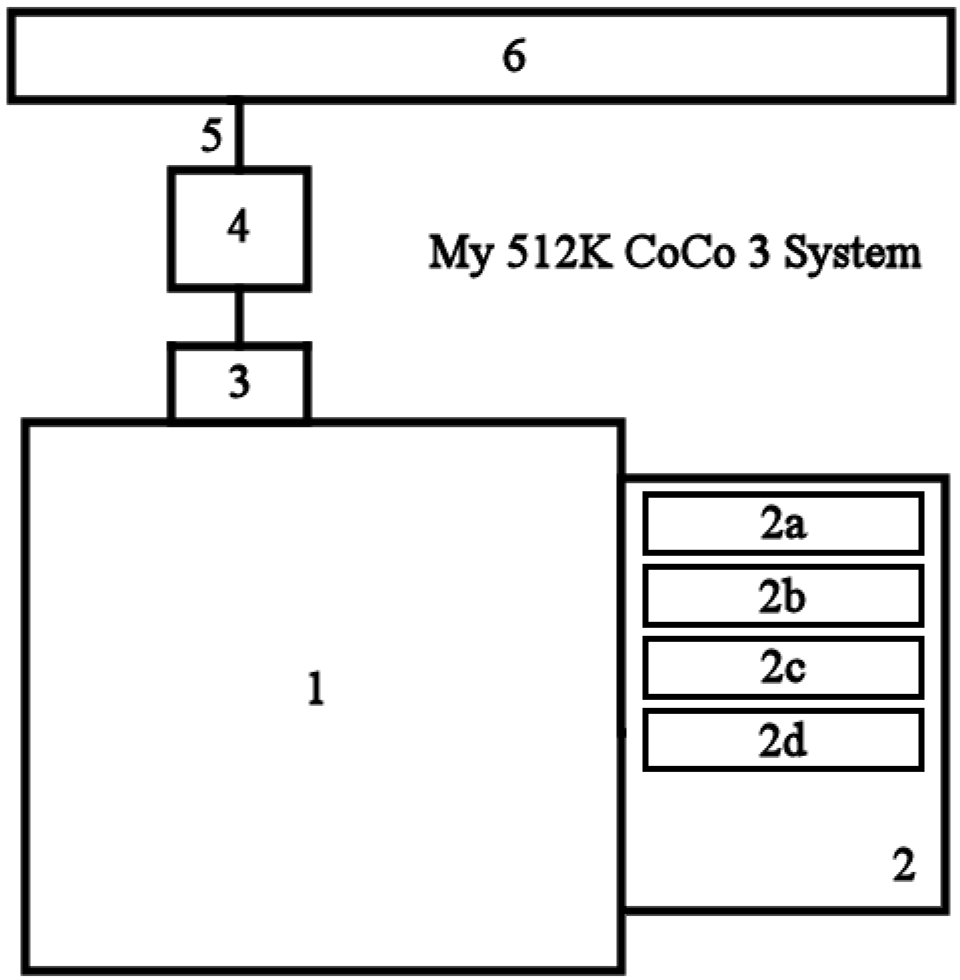
1. 128K CoCo 3, Model Number 26-3334, Serial Number 1110785
2. Mega-Mini MPI
    - 2a. CoCo SDC, running SDC-DOS 1.6 CC3
    - 2b. RS-232 Pak
    - 2c. Empty slot
    - 2d. Empty slot
3. CoCo Switch-a-Roo*
4. TNP SCART to HDMI Converter
5. HDMI Cable
6. RCA 19” VGA/HDMI Monitor, Model RT1970

*Not Shown: The Switch-a-Roo is powered from a ByEasy powered 7-port USB 2.0 hub.

=====

Appendix D: My 512K CoCo 3 System

As of 2021/09/08, my physical 512K CoCo 3 System is depicted in the block diagram below.



See following page for System Legend.

System Legend:

1. 512K CoCo 3, Model Number 26-3334, Serial Number 1037654
2. Mega-Mini MPI
    - 2a. CoCo SDC, running SDC-DOS 1.6 CC3
    - 2b. RS-232 Pak
    - 2c. Empty slot
    - 2d. Empty slot
3. CoCo Switch-a-Roo*
4. TNP SCART to HDMI Converter
5. HDMI Cable
6. RCA 19” VGA/HDMI Monitor, Model RT1970

*Not Shown: The Switch-a-Roo is powered from a ByEasy powered 7-port USB 2.0 hub.

=====

Appendix E: My CoCo Philosophy

The CoCo community enjoys a great diversity of interests.

Some choose to concentrate on hardware innovations and modifications such as interfacing with VGA and HDMI monitors, SD Card data storage, and 104-key keyboards. This interest is at least partly born of necessity, since composite monitors, floppy diskettes, and CoCo spare parts are no longer manufactured and are in increasingly short supply.

Others concentrate on expanding the software horizons of the CoCo 3, using NitrOS-9 and other operating systems to make the multitasking CoCo behave ever closer to modern Windows, Mac, and Linux machines.

Still others are devoted to emulating the CoCo on other platforms by developing emulators such as VCC, OVCC, MAME, and XRoar.

And some just love retro gaming.

My personal interest is twofold:

1. To see VCC increasingly used as a learning tool for budding software developers.

2. To see just how much I can cram into a 64K CoCo 2.

First, VCC: Today's Grade School, Junior High, and High School students have a wealth of available learning tools. Micro-bits, Arduinos, and Raspberry Pi supermicro devices provide highly affordable entry-level introductions to computer programming and interfacing. Maker-Spaces and Innovation Centers in our schools and libraries help foster growth and experience.

But these devices do have limitations. Even these simple(?) computers can have rather steep learning curves, and their low initial cost can quickly expand as new peripherals and experimental equipment and supplies are added.

VCC is free, and can be used on any Windows computer: just download it, install it, and it runs. If you don't own a Windows computer, your school, library, or a friend probably does. The included BASIC language is easy to learn and can readily serve as a stepping-stone towards more complex programming languages. (And, no, learning structured programming does not require a language that enforces structure. In fact, I think learning to structure your programs is actually more effective when you do so on your own.)

I prefer VCC to the other emulators for these purposes because its setup is trivial: Again, just download it, install it, and it runs. OVCC, MAME, and XRoar have their advantages, but ease of setup is not one of them. Even with their available Windows binary packages, they require pre-installation of other bits and pieces of software before they can be downloaded, installed, and run. This may not be a major problem for a reasonably adept aficionado, but it forms a significant barrier for the newbie. And, it’s the newbie whom we’re trying to reach, interest, and encourage here; the newbie who may not yet recognize even the tiniest awakening of interest in things computational.

But, for these purposes, VCC has one glaring weakness: its instruction manual is woefully terse. I would like to see VCC bundled with a selection of tutorials, manuals, and examples suited to guiding even the most newbie of newbies into the wonders of computing.

Second, The Stuffed CoCo: I’m simply fascinated by the challenge of seeing how much functional capability I can sandwich into the nooks and crannies of the 64K space. Whether it’s working in the available RAM left by the 32K ROM and the dedicated RAM that supports that ROM, or whether it’s jumping right into ALLRAM mode and just filling the entire 64K to near-overflowing; it’s an investigative gauntlet which goes right to the heart of my enchantment with puzzles in general.

It’s great fun!

M.D.J. 2021/08/29

=====

Appendix F: New BDS Software License

This New Software License applies to all software found on the BDS Software site, and supersedes all previous copyright notices and licensing provisions which may appear in the software itself or in any documentation therefor.

All software which has previously been placed in the public domain remains in the public domain.

All other software, programs, experiments and reports, documentation, and any other material on this site (other than that attributed to outside sources) is hereby copyright © 2018 (or later if so marked) by M. David Johnson.

All software, documentation, and other information on the BDS Software site is available for you to freely download without cost.

Whether you downloaded such items directly from this site, or you obtained them by any other means, you are hereby licensed to copy them, to sell or give away such copies, to use them, and to excerpt from them, in any way whatsoever, so long as nothing you do with them would denigrate the name of our Lord and Savior, Jesus Christ.

I make absolutely no warranty whatsoever for any of these items. You use them entirely at your own risk.

If they don't work for you, I commiserate.

If they crash your system, I sympathize.

But I accept no responsibility whatsoever for any such consequences. Under no circumstances will BDS Software or M. David Johnson be liable for any negative results of any kind which you may experience from downloading or using these items.

BDS Software's former mail address at P.O. Box 485 in Glenview, IL is no longer valid. Any mail sent to that address will be rejected by the U.S. Postal Service. See my Contact page.

M.D.J. 2018/06/08

=====

Works Cited

"Bare-Metal Programming." *Techopedia*. Web. https://www.techopedia.com/definition/3745/bare-metal-programming . Last Accessed 2021/09/14.

Getting Started With Extended Color BASIC. Fort Worth, TX: Radio Shack, 1983. Print.

Manchester, William and Reid, Paul. *Defender of the Realm 1940-1965*. New York: Little, Brown and Company, 2012. Print. The Last Lion.

[MDJ01] Johnson, M. David. *Key Codes and VIDRAM*. Glenview, IL: BDS Software, 2021. Web. http://www.bds-soft.com/cocoPapers.php . Last Accessed 2021/09/29.

[MDJ02] Johnson, M. David. *Towards a VCC Bundle*. Glenview, IL: BDS Software, 2021. Web. http://www.bds-soft.com/cocoPapers.php . Last Accessed 2021/09/29.

Microsoft. *Disk EDTASM+ 01.00.00*. Fort Worth, TX: Radio Shack, 1983. Print.

Perotti, James and Perotti, Victor. "Assembly 101." *Hot CoCo*, May 1985, 68. Peterborough, NH: CW Communications, 1985. Print.

Snider, Ed. "Assembly Programming." *The Zippster Zone*. Web. https://thezippsterzone.com/programming/ . Last Accessed 2021/09/10.

Warren, Carl. *MC6809 Cookbook, The*. Blue Ridge Summit, PA: TAB Books, 1980. Print.

=====

END